Miguel Alexandre Gonçalves Lourenço

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

mForester – Sistema Móvel de Monitorização, Registo e Comunicação para cenários de Incêndio Florestal

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores

Orientador: João Pedro Oliveira, Professor Auxiliar, FCT-UNL

Júri:

Presidente: Doutor André Teixeira Bento Damas Mora - FCT/UNL

Vogais: Doutor Luís Augusto Bica Gomes de Oliveira - FCT/UNL (Arguente)

Doutor João Pedro Abreu de Oliveira – FCT/UNL (Orientador)

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outubro, 2019

mForester – Sistema Móvel de Monitorização, Registo e Comunicação para cenários de Incêndio Florestal

À minha família...

Agradecimentos

Quero começar por agradecer ao Professor João Pedro Oliveira pela oportunidade que me concedeu em realizar esta dissertação num tema que poderá trazer contributos significativos na melhoria de um problema de extrema importância para a sociedade portuguesa.

À Faculdade, por me proporcionar uma formação de qualidade que foca não só as competências técnicas mas também as *soft-skils,* permitindo preparar os seus alunos para o mercado de trabalho.

Aos meus grandes amigos, André, João e Caique, por toda a amizade e persistência em partilharem momentos comigo em alturas que dei mais atenção à faculdade.

À Equipa d'África e às suas pessoas, que me ensinaram o sentido de ser Missionário e estar aberto a ajudar os outros pondo os meus dons a render. Também pela oportunidade que me deram em missionar em Moçambique e de lá trazer tantas aprendizagens para a vida.

Um agradecimento especial ao meu pai, Jorge, e à minha mãe, Alexandrina por todo o empenho e sacrifício que têm tido ao longo da vida para me proporcionar uma formação de qualidade e um ambiente em que possa crescer enquanto pessoa e profissional. À minha irmã, Sofia, por todo o apoio que me tem dado não só durante o curso mas também ao longo da vida, pela grande amizade e companheirismo. À minha namorada, Margarida, por todo o amor, cuidado e carinho que me tem dado nesta última fase do curso que me deu força para continuar e enfrentar as adversidades. Ao meu cunhado, Fábio, pela grande amizade que temos criado ao longo destes anos.

Aos meus colegas, por todas as vivências partilhadas ao longo destes anos e pela partilha de conhecimentos. Guardo com carinho todas as recordações a

começar pela entrada no curso em que fui recebido com grande hospitalidade e todos os momentos de convívio, trabalhos de grupo, etc. Um especial agradecimento aos meus colegas Tiago Carrasqueira, João Pires, Sebastião Pedrosa e Miguel Loureiro por todos os momentos partilhados quer em trabalho quer em convívio.

E por último, aos meus professores por toda a partilha de conhecimento ao longo destes anos que não só me permitiram adquirir novos conhecimentos como também me estimularam o pensamento crítico.

Resumo

Os incêndios florestais são uma realidade que todos os anos assola Portugal. Nos últimos 10 anos arderam mais de 1,000,000 de hectares e as previsões são que as alterações climáticas potenciem cada vez mais no futuro fenómenos extremos que conduzam a que os anos mais desastrosos se tornem cada vez mais frequentes.

Uma análise ao sistema de combate aos incêndios florestais em Portugal revela uma dependência de meios ainda muito rudimentares para o planeamento do combate face às evoluções tecnológicas do presente e políticas de combate cimentadas sobretudo na aquisição de mais meios.

Face a esta necessidade crescente de integrar meios tecnológicos no combate, foi desenvolvido um sistema de apoio a decisão para o combate aos incêndios florestais baseado em tecnologias de baixo-custo e que ponha à disposição dos comandantes no teatro de operações mais dados de forma a permitir decisões mais céleres e informadas.

Palavras-chave: Incêndios florestais, Sistema de apoio a decisão, baixo-custo

Abstract

Forest fires are a reality that every year plagues Portugal. Over the past 10 years, more than 1,000,000 hectares have burned and the predictions are that climate change will increasingly potentiate extreme events that lead to disastrous years becoming more frequent.

An analysis of the Forest fire combat system in Portugal reveals a dependence on still very rudimentary means for the planning of the firefighting considering the technological evolutions of the present and policies of firefighting cemented mainly in the acquisition of more resources.

Given the growing need to integrate technological means into the firefighting, a decision support system was developed for forest fire-fighting based on *low-cost* technologies and to make available to commanders in the theater of operations more data to enable faster and more informed decisions.

Keywords: Forest Fire, Decision Support System, *low-cost*

Conteúdo

Lista de figuras

Lista de Tabelas

Siglas

API – *Application Programming Interface*

ATA – *Ataque Ampliado*

ATI – *Ataque Inicial*

BLE - *Bluetooth Low Energy*

CB – *Corpos de Bombeiros*

CDOS - *Comando Distrital de Operações de Socorro*

CMA – *Centro de Meios Aéreos*

COPAR – *Coordenador de Operações Aéreas*

COS – *Comandante de Operações de Socorro*

GPIO - *General Purpose Input/Output*

HEATI – *Helicóptero de Ataque Inicial*

HERAC – *Helicóptero de Reconhecimento, Avaliação e Coordenação*

IoT - *Internet of things*

LPWAN – *Low Power Wide Area Network*

PCO – *Posto de Comando Operacional*

PWM - *Pulse Width Modulation*

RAM – *Random Access Memory*

REPC – *Rede Estratégica de Proteção Civil*

ROB – *Rede Operacional dos Bombeiros*

ROM - *Read-only Memory*

SDK – *Software Development Kit*

SIG – *Sistema de Informação Geográfica*

SIRESP – *Sistema Integrado de Redes de Emergência e Segurança de Portugal*

SoC - *System on Chip*

TO – *Teatro de Operações*

UART - *Universal asynchronous receiver/transmitter*

ULP - *Ultra-lower-power co-processor*

WiFi - *Wireless Fidelity*

WSN – *Wireless Sensor Network*

1

Introdução

Este capítulo iniciar-se-à com uma breve introdução ao problema dos fogos em Portugal, apresentando algumas estatísticas e problemas que motivarão a realização desta dissertação. Será também apresentado o problema a ser dado resposta e os objetivos a que esta dissertação se propõe assim como a organização do documento.

1.1. O problema dos fogos em Portugal

A incidência dos fogos em Portugal tem aumentado na última década com fogos de grandes dimensões a serem mais frequentes. O ano de 2017 foi o que apresentou a maior área ardida dos últimos 10 anos com um total de 539,921 hectares ardidos em 21,006 incêndios rurais, contando também com a perda de 67 vidas humanas. A seguir a este, só mesmo o de 2003 em que arderam 471,750 hectares em 28,087 incêndios rurais [1].

A forte incidência de incêndios florestais em Portugal deve-se a um conjunto de fatores. A começar pelo clima, que em Portugal, no resto da bacia Mediterrânica e noutras regiões do mundo com climas semelhantes, potencia um

ambiente favorável ao surgimento de incêndios florestais devido à alternância de uma estação chuvosa com períodos secos e de calor extremo que promovem uma elevada produção de biomassa vegetal culminando em grandes incêndios no verão [2]. Este fator tem sido gradualmente intensificado pelas alterações climáticas, e também ao envelhecimento e despovoamento da população rural aliado às fracas políticas de prevenção reforçadas pelo abandono de atividades agrícolas e de pastorícia em zonas do interior que contribuíam para a gestão de combustível.

1.2. Enquadramento do problema

O sistema de combate aos fogos atual centraliza grande parte da informação no posto de comando. No Teatro de Operações (TO), os comandantes recorrem maioritariamente a cartas militares não havendo um suporte informático para o registo e planeamento do combate e acesso a dados históricos e correntes da zona da ocorrência, como as caraterísticas do terreno, os acessos por estrada ou os pontos de água nas redondezas, estando muito dependentes da ligação com o posto de comando que detém essas informações.

O suporte com ferramentas tecnológicas e conhecimento científico torna-se portanto uma mais valia no apoio à decisão, tal como foi sugerido por Mark Beighley e A. C. Hyde no seu relatório "Gestão dos Incêndios Florestais em Portugal numa Nova Era Avaliação dos Riscos de Incêndio, Recursos e Reformas" em que apontavam "Observa-se também, embora de um modo muito menos visível, uma falta de influência e de aplicação prática da investigação científica e das tecnologias emergentes na gestão dos recursos naturais, apoio à tomada de decisões e serviços de previsão de incêndios florestais. Apesar de existirem, já há vários anos, oportunidades de correção destas deficiências, foram feitos poucos progressos, o que sugere a necessidade de uma abordagem mais agressiva. " [3].

Desta forma, dotando os comandantes no TO de mais informações sobre o estado do incêndio, é possível reduzir o tempo de planeamento e iniciar-se o combate mais rapidamente fundado em mais dados operacionais.

Além disso, é também importante fazer o registo da ocorrência para que no futuro se possa estimar o comportamento do fogo numa determinada localização em que parâmetros como a topografia do terreno e os combustíveis poderão ser semelhantes entre ocorrências e, comsequentemente, melhorar estratégias de combate.

Existem atualmente algumas soluções que serão analisadas no decorrer desta dissertação e será, posteriormente, proposta uma outra que vise não substituir, mas sim complementar as soluções existentes procurando preencher lacunas que estas apresentem.

1.3. Enquadramento com projeto e contribuições principais

Esta dissertação encontra-se integrada no projecto FORESTER - PCIF / SSI / 0102 / 2017 - Rede de sensores combinada com modelação da propagação do fogo integrado num sistema de apoio à decisão para o combate a incêndios florestais, com o apoio financeiro da componente do Orçamento de Estado da FCT/MCTES e cujo objetivo é o da implementação de uma aplicação em software GIS que combina inputs para sistema de apoio à decisão

1.4. Objetivos da Dissertação

Com base no apresentado na seção anterior surge então a seguinte questão:

- "Como é que, recorrendo às potencialidades das ferramentas tecnológicas, nomeadamente a plataformas de *hardware* de baixo custo e sistemas de informação geográfica, poder-se-à desenvolver um sistema de apoio a decisão para o combate a incêndios florestais?"

Assim sendo, o objetivo principal desta dissertação é o da concepção, projeto, implementação e teste da primeira *release* de um equipamento de fácil utilização destinado à monitorização, registo e comunicação em cenários de Incêndio Florestal.

Este equipamento será constituído por um dispositivo móvel sensorial, doravante denominado *device*, que fará a sensorização e transmissão de parâmetros atmosféricos de interesse, e de uma aplicação para *tablet* que permitirá aceder a estes dados e irá conter um conjunto de funcionalidades típicas de um sistema de informação geográfica que darão apoio ao combate aos incêndios florestais.

Algumas caraterísticas serão essenciais neste equipamento:

- Aplicação intuitiva e com boa *UX* (*User Experience*).
- *Device* compacto e de fácil configuração.
- Possibilidade de trabalhar *offline*
- Funcionalidades de um Sistema de Informação Geográfica
- Baixo custo

1.5. Organização do documento

Este documento encontra-se organizado em três capítulos:

1. **Introdução**, que começa por introduzir o problema dos fogos em Portugal e apresenta o enquadramento do problema, objetivos e visão da dissertação, de maneira a clarificar o leitor acerca do problema a que se propõe a resolver e do projeto em que se enquadra;
2. **Tecnologias de suporte a sistemas de apoio a decisão em ambiente de incêndio florestal**, que começa por abordar a atual metodologia de combate aos fogos em Portugal, apresenta, numa abordagem *top-down*, exemplos de sistemas de apoio a decisão que agreguem um conjunto alargado de dados para potenciar decisões informadas, seguido de componentes que integrem os sistemas de apoio a decisão e que permitam a deteção de incêndios florestais, terminando com exemplos de plataformas de *hardware* e tecnologias de comunicação que poderão ser usadas para dar resposta ao problema.
3. **Trabalho desenvolvido**, onde são apresentados os requisitos que o sistema a ser desenvolvido deve cumprir e detalhados todos os passos de implementação das várias componentes do sistema.
4. **Conclusão e trabalho futuro -** onde é feita uma síntese do trabalho desenvolvido no âmbito desta dissertação, referindo os requisitos que foram implementados e as limitações do sistema, concluindo com sugestões de trabalho futuro com vista a melhorar o que foi feito e implementar novas funcionalidades.

2

Tecnologias de suporte a sistemas de apoio a decisão em ambiente de incêndio florestal

Este capítulo iniciará com a análise à metodologia de combate aos fogos em Portugal apresentando as entidades intervenientes e as suas interações. Posteriormente, serão apresentados exemplos de componentes de vários níveis dum sistema numa abordagem *top-down* passando dos componentes de mais alto nível como Sistemas de Apoio a Decisão (SAD), para os de mais baixo nível como os protocolos de comunicação pesquisados. A Figura 1 representa num diagrama a sequência da abordagem utilizada.

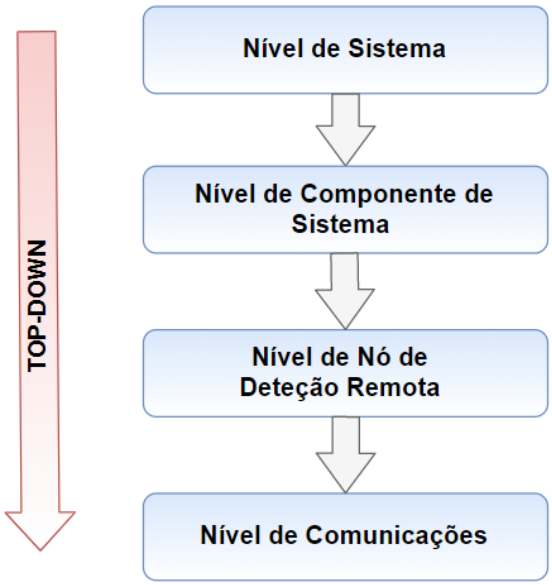

Figura 1 - Diagrama da abordagem utilizada neste capítulo

Inicialmente serão apresentados e discutidos alguns dos atuais sistemas de apoio ao combate a incêndios florestais concluindo-se quanto à sua adequação face ao problema que esta dissertação procura responder.

No final serão apresentadas alguns exemplos de plataforma de *hardware* e tecnologias de transmissão de dados que poderão ser interessantes para soluções de *remote sensing* aplicadas ao problema em causa.

2.1. Breve perspetiva sobre o combate aos incêndios florestais em Portugal

O combate aos incêndios florestais inicia-se com o alerta que poderá ter origem no 112, postos de vigia da GNR, alerta das populações ou das câmeras de vigilância dos Comandos Distritais de Operações de Socorro (CDOS). Após o alerta ser dado, é iniciada a fase de Ataque Inicial (ATI) em que é feita a triangulação do incêndio com a saída das viaturas de primeira intervenção dos três Corpos de Bombeiros (CB) mais próximos do local de incêndio tendo em vista uma resposta célere e eficaz.

Em simultâneo com os meios terrestres, poderão ser acionados pelo CDOS, a partir dos Centros de Meios Aéreos (CMA), os Helicóptero de Ataque Inicial (HEATI), com a sua equipa de 5 (cinco), 8 (oito) ou 12 (doze) operacionais caso a ocorrência se encontre a uma distância inferior a 40 km.

No caso de estarem envolvidos mais de 2 (dois) meios aéreos no TO, deve ser nomeado um Coordenador de Operações Aéreas (COPAR) que fará a ligação entre os meios aéreos e o Posto de Comando (PCO) e garantirá o cumprimento dos objetivos táticos atribuídos aos meios aéreos. Se estiverem a operar mais de 4 (quatro) meios aéreos, é ativado um Helicóptero de Reconhecimento, Avaliação e Coordenação (HERAC) que fará a sua orientação no TO.

Se 90 (noventa) minutos após a saída do primeiro meio de intervenção o incêndio ainda não ter sido dado como dominado pelo Comandante de Operações de Socorro (COS), é dado início à fase de Ataque Ampliado (ATA).

Assim que uma operação passa à fase ATA é necessário assegurar o reforço dos meios operacionais no TO a partir de CB locais e nas proximidades. Nesta fase, poderão também ser acionados os Helicópteros de Ataque Ampliado (HE-ATA) que operam num raio de 70 km.

Em ambas as fases são empregues métodos de combate combinado e/ou indireto com recurso a ferramentas manuais como p.e. roçadouras, abafadores e ancinhos, assim como tratores agrícolas e máquinas de rasto aliados à utilização coordenada de fogo de supressão sobre a responsabilidade de um técnico especializado e o COS, sendo esta ação aprovada pela estrutura de comando e registada a ocorrência na fita do tempo [4][5].

A comunicação entre as várias forças operacionais no TO é feita essencialmente por três sistemas de comunicações: os sistemas em VHF/AM (Banda Aeronáutica) utilizados para a comunicação com os meios aéreos; os sistemas em VHF/FM (Banda Alta), em particular a Rede Estratégica de Proteção Civil (REPC) e a Rede Operacional dos Bombeiros (ROB); e o Sistema Integrado de Redes de Emergência e Segurança de Portugal (SIRESP). Os sistemas ROB e REPC, em conjunto com as redes móveis convencionais, podem também ser usados como sistemas redundantes em caso de limitações de outros sistemas como aconteceu no incêndio de Pedrogão Grande em 2017 em que a Estação Base SIRESP entrou em modo local não permitindo a comunicação entre o PCO e alguns operacionais no TO [6].

Constata-se que o atual sistema funciona, aliado também a ferramentas tecnológicas que prestam apoio à decisão ao fornecer informação geográfica, como é o caso do geoMAI, georreferenciação dos meios operacionais no caso do SIRESP GL ou até na previsão de comportamento do fogo com o Sistema de Simulação de Propagação de Incêndios. No entanto, é necessário cada vez mais fazer convergir as várias áreas afetas ao fogo com a resposta operacional no terreno integrando conhecimento científico, tal como foi no sugerido na "Resolução do

Conselho de Ministros n.º 159/2017" em que se mencionou o relatório realizado pela Comissão Técnica Independente (CTI) sobre os Incêndios ocorridos em Pedrogão Grande, Castanheira de Pera, Ansião, Alvaiázere, Figueiró dos Vinhos, Arganil, Góis, Penela, Pampilhosa da Serra, Oleiros e Sertã, entre 17 e 24 de junho de 2017 apontando a "necessidade de promover a investigação científica e a inovação, integrando avanços emergentes da ciência e a adaptação e integração de boas práticas identificadas internacionalmente, nomeadamente nas áreas da meteorologia, da silvicultura, da gestão do fogo e previsão do seu comportamento" [7] e promover também a criação de soluções com uma maior integração de informação e que disponham de redundância nos sistemas de comunicações.

2.2. Sistemas de apoio a decisão para o combate aos incêndios florestais

Os Sistemas de Apoio ao Combate aos incêndios florestais atualmente, em Portugal, baseiam-se muito em SIG presentes nos postos de comando [44]. Estes sistemas comportam todas as fases desde a aquisição, armazenamento, análise e visualização de dados de cariz espacial [45].

2.2.1. Sistemas de Informação Geográfica (SIG)

Os SIG são ferramentas computacionais que permitem analisar, guardar, manipular e visualizar informação geográfica num mapa auxiliando assim ao seu enquadramento espacial facilitando a tomada de decisões.

Num SIG, os dados são guardados como *rasters* (grelhas) ou vetores. Na forma de *rasters,* a informação é disposta numa grelha e pode ser classificada de duas formas:

- *Rasters* contínuos – células numa grelha com dados de mudança gradual. Por exemplo, dados de parâmetros atmosféricos como

temperatura ou dados de *digital elevation models* (DEM) que são modelos de elevação digitais da superfície de um terreno.

- *Rasters* discretos – quando existem dados que podem ser categorizados. Representa essencialmente objetos que têm limites, por exemplo um lago.

Já os vetores, não são constituídos por uma grelha de píxeis. Podem antes ser, por exemplo, pontos, linhas e polígonos usados para representar estradas, caminhos férreos e fronteiras entre regiões.

No contexto do apoio ao combate de fogos florestais, os SIG podem ser usados, por exemplo, para visualizar imagens de satélite duma zona em que esteja a ocorrer um incêndio e com base na rede viária, informação de pontos de água nas redondezas e localização de quartéis de bombeiros, fazer o planeamento do combate ao incêndio.

Também numa fase de simulação do comportamento do fogo numa determinada região, é possível com estas ferramentas construir mapas de risco dos impactos de um incêndio em função dos tipos de vegetação da zona e dos dados climatéricos [45].

Por outro lado, depois de um incêndio será benéfico para planeamento futuro, guardar dados geográficos e informação geo-espacial de forma a melhorar estratégias de combate.

Os SIG gozam duma grande adoção havendo na literatura múltiplos exemplos da sua utilização. Um exemplo de aplicação em Portugal foi proposto por Carvalho, F. [47] em que foi desenvolvido um SIG para apoiar o combate aos fogos florestais no concelho de Águeda que visa complementar o dispositivo existente, baseado essencialmente em informação empírica, com informação geográfica, e que facilite a comunicação entre os vários agentes intervenientes no combate ao fogo. O sistema é constituído por três módulos distintos: uma aplicação *WebSIG,* que será o módulo principal e permitirá visualizar os meios terrestres; uma aplicação para dispositivos móveis, para os meios terrestres poderem introduzir a sua localização periodicamente de forma automática; e uma

aplicação para dispositivos móveis para os meios aéreos, que permitirá a estes meios encontrar locais para o abastecimento de água.

Outro exemplo de aplicação é o sistema *MacFire,* desenvolvido pelo município de Mação em conjunto com a empresa *ESRI*. O MacFire é um sistema de apoio a decisão que permite agregar informação sobre cartografia militar, cartas de risco de incêndio, hortofotogramas e posição das frentes de fogo permitindo assim, segundo Engº António Louro responsável pela Proteção Civil Municipal de Mação, auxiliar na "(…) escolha do ponto adequado para ataque a um ponto de incêndio, ou a determinação do caminho para lá chegar, funcionando como uma célula de planeamento das intervenções a realizar." [46]. Aliado a isto, o sistema também dispõe de georreferenciação de todos os meios no terreno, bem como a posição das frentes de fogo e a dimensão da área ardida. A integração desta informação em conjunto com informação histórica que o sistema disponibiliza, permite prever o eventual comportamento do fogo, contribuindo para uma estratégia de combate mais eficaz [46] [49].

A georreferenciação dos meios no TO é uma medida de extrema importância que foi sugerida pelo investigador Domingos Xavier Viegas no relatório "O Complexo de Incêndios de Pedrógão Grande e concelhos limítrofes, iniciado a 17 de junho de 2017" sobre Pedrógão Grande onde salientou que "É importante fiscalizar esta medida para evitar situações que se encontram frequentemente, de meios que existem, mas que não são efetivamente ativados, ou os seus dados não estão acessíveis." [48]. Desta forma, ao saber-se a todo o momento os meios disponíveis no terreno, é possível um melhor planeamento e uma maior articulação entre os mesmos permitindo também a sua melhor coordenação por parte dos postos de comando.

O sistema tem um custo de 125 mil euros e é constituído por uma carrinha adaptada com vários equipamentos de comunicação, *software* e seis monitores que disponibilizam toda a informação em tempo real do posicionamento dos bombeiros, acessos viários, localização dos pontos de água e dimensões do fogo, servindo então como um posto de comando móvel que se pode deslocar para o TO.

Este sistema já tem sido testado em cenários de incêndio, sendo o incêndio de Monchique um exemplo de atuação, e pretende-se expandir para todo o concelho de Santarém com a colaboração das comunidades intermunicipais do Médio Tejo e da Lezíria do Tejo [46] [49].

2.2.1.1. Plataforma PGIR

A *Plataforma Integrada de Gestão de Riscos (PGIR)* é uma plataforma SIG coordenada pela *Esri Portugal* e com o contributo do *Sistema Científico-Tecnológico Nacional*, *Associação para o Desenvolvimento da Aerodinâmica Industrial (ADAI)* e *Departamento de Engenharia Civil* da *Faculdade de Engenharia* da *Universidade do Porto* que permite caraterizar vulnerabilidades, apoiar na gestão de riscos inerentes a desastres naturais e conceber modelos de prevenção e de resposta eficaz e integrada, pelos Serviços de Segurança e de Proteção Civil, orientada para cenários de Incêndios Florestais, Cheias e Inundações.

Nos cenários de Incêndios Florestais, o *PGIR* permite obter áreas de risco de incêndio, campos de vento, a taxa de propagação e comprimento da chama, intensidade da reação e da linha de fogo e as áreas queimadas.

Na base do seu funcionamento estão presentes técnicas científicas de modelação hidrológico-hidráulica e de propagação de fogos florestais que permitem auxiliar as autoridades competentes a deterem uma visão espacial dos dados inerentes a estes cenários e assim auxiliar a tomada de decisão [50].

2.2.1.2. Sistema geoMAI

O *geoMAI* é um SIG do *Ministério da Administração Interna (MAI)* baseado em tecnologia da *ESRI* que pretende ser uma plataforma interoperável e transversal para a partilha e consumo de informação entre as várias forças e serviços de segurança do *MAI,* possibilitando-lhes de fazer as suas análises suportadas na informação existente e desenvolver as suas próprias aplicações traduzindo-se assim num apoio à tomada de decisão.

Esta plataforma pretende suportar a ação desde a fase de planeamento até à área operacional garantindo que a informação é segura, atualizada e de acesso restrito e credenciado.

Também o desempenho e disponibilidade são assegurados através duma solução robusta que se encontra enquadrada na *Rede Nacional de Segurança Interna (RNSI),* que garante por si só robustez. No caso de perda de conetividade durante uma emergência, o *geoMAI* garante a capacidade tecnológica para se poder continuar a trabalhar não comprometendo assim a resposta operacional.

O *geoMAI* integra numa única base dados uma panóplia de dados operacionais georreferenciados de diversas fontes como p.e. informação cartográfica, imagens de satélite, dados meteorológicos, entre outros, permitindo um conjunto de funcionalidades desde a interligação da informação e utilização de algoritmos de análise espacial, à consulta, visualização e localização de conteúdo da base de dados geográfica, complementando com ferramentas de *business intelligence* [51] [53].

Com a criação do *geoMAI,* melhorou-se a interoperabilidade dos sistemas das várias forças e serviços de segurança do *MAI,* contribuindo para uma menor dispersão de informação o que se traduz numa maior integração entre as forças potenciando a gestão e racionalização de recursos assim como a criação de estatísticas harmonizadas e obtenção de informação geográfica adequada à missão de cada uma das forças [51] [52] .

Os principais benefícios que esta plataforma traz são uma maior eficiência na adequação dos meios alocados a ocorrências, uma melhoria da capacidade de resposta operacional e uma melhoria na qualidade da informação necessária à atividade operacional da administração interna, o que se traduz positivamente na segurança pública [51] [53].

2.2.1.3. Sistema EFFIS

European Forest Fire Information System (EFFIS) é um SIG modular que surgiu em 2000 e que desde 2015 está inserido na componente de *"Early Warning"*

do *"Copernicus Emergency Management Service"*, um dos seis serviços disponibilizados pelo programa *Copernicus* [54] [55].

Este sistema foi desenvolvido pela Comissão Europeia em colaboração com Administrações Nacionais na área do fogo e desde 1998 conta com o apoio de uma rede de especialistas de 43 países da Europa, Médio Oriente e Norte de África que constituem o chamado *"Expert Group on Forest Fires"* [56] [57].

O *EFFIS* permite monitorizar a atividade de fogos florestais quase em tempo-real assim como fornecer dados históricos sobre o seu comportamento nas regiões da Europa, Médio Oriente e Norte de África permitindo auxiliar a gestão do combate aos incêndios tanto a nível nacional como a nível regional.

Para tal, este sistema conta com uma série de módulos que abrangem o ciclo inteiro do fogo desde as condições prévias aos incêndios até à avaliação dos danos pós-incêndio.

Na base do *EFFIS* encontra-se uma base de dados denominada *"European Fire Database"* que inclui informação detalhada de incêndios ocorridos em países integrantes deste sistema e que conta atualmente com quase 2 milhões de registos fornecidos por 22 países que são valiosos para que investigadores e decisores políticos possam fazer uma análise temporal dos incidentes [57] [58].

Para além destes módulos, o *EFFIS* conta também com outro módulo designado *"Fire News"* que georreferencia todas as notícias relacionadas com incêndios florestais disponíveis na Internet em qualquer uma das línguas faladas na Europa [57].

Desde que ficou operacional em 2000, o *EFFIS* tem contado com melhorias significativas ao ser complementado com novos módulos para o processamento e disponibilização de dados resultantes da monitorização de fogos florestais. Estes desenvolvimentos têm beneficiado da forte evolução tecnológica dos sistemas de informação e comunicação em particular dos sistemas de sensorização remotos e dos SIG.

2.2.2. Outros Sistemas de Apoio a Decisão

2.2.2.1. Sistema SADO

O *Sistema de Apoio à Decisão Operacional (SADO)* é um sistema desenvolvido pela empresa *Indra* para a *Autoridade Nacional de Proteção Civil (ANPC)* que vem substituir o antigo sistema designado *Proteção Civil Gestão de Ocorrências (PCGO)* e que tem como objetivo permitir uma resposta mais célere e eficaz das autoridades ao assegurar uma infrastrutura de partilha de dados entre os Agentes da Proteção Civil graças a uma integração de informação mais robusta e alargada.

Esta integração da informação permite além disso a melhoria na tomada de decisões e na sua rapidez contribuindo também para um aumento na eficácia do planeamento, coordenação e execução de atividades e gestão de meios de proteção e socorro.

O *SADO* também assegura uma maior fiabilidade ao garantir a sua permanente operacionalidade e reúne os dados operacionais da *ANPC* na sua plataforma permitindo assim centralizar toda a informação estatística e facilitar a sua análise e seu fornecimento a entidades externas.

Neste sistema estão incorporados vários subsistemas sendo um dos quais o *Subsistema Integrado de Gestão de Meios (SIGM)* que permite: listar e fazer a gestão operacional dos meios e recursos do *SADO*; fazer a a gestão das entidades envolvidas nas ocorrências; gerir os contratos associados aos meios aéreos; fazer a gestão de *stocks* de combustíveis, das necessidades dos *CB* e de outros recursos necessários.

Esta plataforma foi desenvolvida recorrendo à *framework* ASP.NET MVC 2.0 e a sua interoperabilidade com outros sistemas é conseguida graças à utilização da *Windows Communication Foundation (WCF).* Além disso encontra-se integrada nas infraestruturas de comunicação da *Rede Nacional de Segurança Interna (RNSI)* [59] [60].

Na Figura 2 encontra-se um esquema dos exemplos de SAD analisados no capítulo 2.2. divididos na categoria “SIG” no caso de sistemas baseados em SIG e na categoria “Outros” para outro tipo de sistemas.

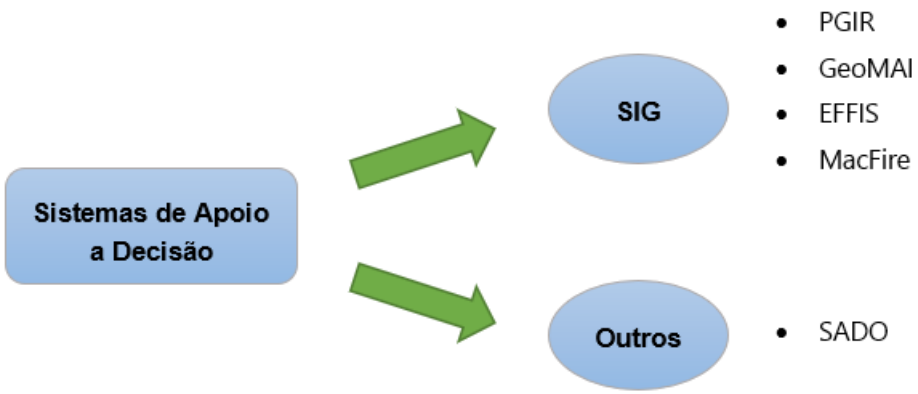

Figura 2 - Exemplos de SAD analisados

2.3. Componentes para sistemas de deteção de incêndios florestais

As redes de sensores denominadas *Wireless Sensor Networks (WSN)* têm-se mostrado uma solução pouco dispendiosa e relativamente eficiente para a deteção e acompanhamento de fogos florestais. Por outro lado, as imagens de satélite têm a grande vantagem de não necessitar de montar uma infrastrutura, uma vez que dispõe de uma rede de satélites existente. Nesta secção são apresentadas ambas as abordagens com exemplos de aplicação.

2.3.1. Wireless Sensor Networks (WSN)

As denominadas *Wireless Sensor Networks* (WSN) são redes de sensores de baixo custo que com o advento da *Internet of Things (IoT)* têm-se tornado cada vez mais populares. Estas redes de sensores têm como finalidade adquirir informação sensorial sobre o seu ambiente, processar essa informação e transmiti-la para outro elemento da rede (p.e uma *gateway* ou *base station*) [31]. Algumas das suas caraterísticas principais são a baixa latência, baixo consumo energético, baixa capacidade de processamento, elevada autonomia e elevada robustez à

falha de nós. Cada nó é, de um modo geral, constituído pelos seguintes módulos [32]:

- Módulo de Energia – responsável pela gestão de energia do nó; constituído tipicamente por uma bateria de média/grande capacidade, reguladores de tensão eficientes e poderá também adotar métodos de *energy harvesting* recorrendo, por exemplo, a painéis fotovoltaicos para aumentar a autonomia do nó.
- Módulo de Sensorização – constituído pelos sensores que medem as grandezas físicas de interesse como temperatura, humidade, entre outros parâmetros, tendo estes uma saída numa escala de tensão, de corrente, ou através duma interface como, por exemplo *I²C*, *UART* ou *RS232*.
- Módulo de Processamento – responsável pelo processamento dos dados sensoriais e sua transmissão para outros nós; constituído tipicamente por um *ADC* (Conversor Analógico-Digital) ou interfaces como *I²C*, *UART* ou *RS232* e um microcontrolador que processe os dados segundo uma sequência de instruções previamente programadas e uma interface com o módulo de comunicação.
- Módulo de Comunicação – este módulo é responsável pela transmissão dos dados para outros nós, ou para *gateways* ou *base stations*; utilizam-se tipicamente protocolos como *Zigbee*, *LoRa*, *Sigfox*, entre outros.
- Módulo de Segurança – eventualmente poderá conter um módulo de segurança responsável por encriptar os dados recorrendo, por exemplo, a especificações como *AES* ou *SHA-2*.

A utilização de *WSN's* para a deteção e apoio ao combate aos fogos florestais traz vantagens uma vez que estes sistemas tendencialmente têm um baixo custo para áreas reduzidas e requerem pouca manutenção, uma vez que gozam de uma elevada autonomia. Têm também a particularidade de serem escaláveis sendo a integração de novos nós em redes do tipo *Star* ou *Mesh* facilmente

conseguida [31]. As suas aplicações passam pela deteção precoce dos incêndios, sendo possível triangular com alguma precisão a sua localização hávendo nós georreferenciados.

No entanto, as desvantagens destes sistemas passam pela sua desadequação para áreas muito substanciais uma vez que o seu custo poderá encarecer significativamente; a sua manutenção que, ainda que pouco frequente, seria morosa em redes de grandes dimensões; a perda de sensores em situações em que um incêndio atingiria diretamente a área onde estão instalados, tanto em fogos de superfície caso a sua instalação fosse a nível do solo ou em fogos de copas no caso em que a sua instalação fosse a nível das árvores; e finalmente o facto de estes serem úteis para a deteção dos incêndios mas a sua utilização exclusiva disponibilizar informação incompleta para apoio à decisão.

As aplicações das *Wireless Sensor Networks* têm sido amplamente estudadas na literatura. A título de exemplo, Abdullah, S. et al. [33] desenvolveram um design de um nó capaz de adquirir parâmetros ambientais e fazer o *upload* dos mesmos no servidor da *Advanced Fire Fighting (AF3)* (projeto que opera sobre o *7th Framework Programme (FP7)*, programa de financiamento *da European Unions Research and Innovation*). Foram testados com fogo ao todo 6 nós que tiveram um comportamento satisfatório e foram resistentes à extinção com água. O sistema desenvolvido é facilmente escalável já que os nós são automaticamente adicionados à rede aquando da sua deteção. No futuro os autores pretendem melhorar a eficiência dos seus sistemas e tamanho dos nós.

Numa outra aplicação, Yingli Zhu, Lingqin Xie e Tingting Yuan [34] desenvolveram um sistema de monitorização para incêndios florestais em tempo-real de baixo consumo energético com base numa *WSN* e tecnologia *GPRS,* que consegue distinguir com precisão vários cenários de incêndios florestais assim como determinar a direção de propagação do fogo. Este sistema tem por base um *SoC CC2531*, um módulo de *GPRS MC55*, um módulo de condicionamento de sinal ligado a um módulo sensorial com um sensor de temperatura e humidade, e um detetor de fumo fotoelétrico que deteta o fumo através da luz infravermelha refletida pelas partículas de fumo que atingem o recetor.

O problema da eficiência energética dos protocolos de encaminhamento (*routing*) foi exposto por V. Devadevan e S. Suresh [35] que compararam os protocolos de *routing DSDV*, *LEACH* e *APTEEN* para a monitorização de fogos florestais, concluindo que o protocolo *APTEEN* é o que permite consumir menos energia ao se dividir a área do incêndio florestal em várias zonas e é o melhor a nível de atraso de entrega de pacotes. Além disso, também concluíram que o protocolo *APTEEN* reduz o número de transmissões e usa parâmetros para transmissão reativa quando necessário. Os autores sugerem que futuramente se inclua informação acerca da localização nos protocolos de *routing* no contexto de fogos florestais, de forma a se poder analisar o consumo energético, perda de pacotes e latência na rede além da questão da segurança e QoS (*Quality of Service*).

2.3.2. Imagens de Satélites

A monitorização de incêndios florestais através de imagens de satélite é maioritariamente usada em fogos de grandes dimensões. Dois exemplos de sensores utilizados são o *Moderate Resolution Imaging Spectroradiometer (MODIS)* presente nos satélites *Terra* e *Aqua* da *NASA* e o *Visible Infrared Imaging Radiometer Suite (VIIRS)* presente no satélite *Suomi NPP*.

O sensor *MODIS* deteta pontos quentes na sua banda térmica com uma resolução de 1000 metros por píxel ao passo que o sensor *VIIRS* deteta com uma resolução de 375 metros por píxel [36]. Esta diferença permite ao *VIIRS* detetar fogos de menores dimensões e de temperatura inferior, que o *MODIS* poderá não conseguir detetar e além disso delinear com maior precisão o perímetro do fogo.

No entanto, o sensor *MODIS* tem uma resolução superior em superfícies que não estejam a arder comparativamente ao *VIIRS*, apresentando nestas circunstâncias uma resolução de 250 metros por píxel superior à do *VIIRS* de 375 metros por píxel [36]. A combinação dos dados de ambos os sensores permite modelar e prever com maior precisão o comportamento do fogo e assim contribuir para uma estratégia de combate mais eficaz.

Um exemplo de aplicação descrito por Maden, U. [37] é o de um sistema de deteção e monitorização de fogos florestais baseado em imagens de satélite desenvolvido através da colaboração entre o *SERVIR-Hindu Kush Himalaya* no *International Centre for Integrated Mountain Development (ICIMOD)* e o Departamento de Florestas do Nepal que permite alertar as populações residentes numa área em que um incêndio seja detetado através de uma mensagem de texto que disponibiliza a localização e a dimensão do incêndio.

O sistema compreende as fases de aquisição de dados, do seu processamento e identificação da localização dos fogos ativos presentes em cada uma das duas passagens que faz diariamente no território deste país. Após fazer a identificação, procede então ao alerta das populações e autoridades locais com um atraso de cerca de 20 minutos e publica numa *web application* a informação que poderá ser visualizada e descarregada por utilizadores registados.

Desta forma, recorrendo às potencialidades dos sensores dos satélites, é possível criar ferramentas de auxílio à gestão do combate aos fogos florestais.

Outra das potencialidades dos sistemas com base em imagens de satélite prende-se à fase posterior aos incêndios de quantificação da área ardida, como foi exemplificado por W. Kiadtikornthaweeyot, C. Sukawattanavijit e A. Rungsipanich em [38] onde apresentam um algoritmo para determinação da área ardida com base no índice *Normalized Burn Rate (NBR)* e em classificação de imagens provenientes do satélite *Landsat-8*. Neste estudo os autores compararam o índice *NBR* antes e depois de um incêndio numa determinada área e fazem também uma comparação entre o algoritmo proposto e o *shapefile* (ficheiro que contém dados geoespaciais) criado através de observações no terreno.

2.3.2.1. Programa *COPERNICUS*

O *Copernicus* é um programa de observação da Terra liderado pela Comissão Europeia e que conta com a parceria de várias entidades entre as quais a Agência Espacial Europeia (ESA), os Estados Membros da UE, a *European Organization for the Exploitation of Meteorological Sattelites (EUMETSAT)*, a *European Centre for Medium-Range Weather Forecasts (ECMWF)*, entre outras e que

sucede ao programa *Global Monitoring for Environment and Security (GMES)*. Este programa tem como objetivo a disponibilização de serviços que permitam acesso a dados precisos e atualizados assim como informação fiável extraída dos mesmos de forma a contribuir para o melhoramento de um conjunto de aplicações das quais se incluem gestão ambiental e urbana, agricultura, saúde, transportes, alterações climáticas, proteção civil, entre outras [39] [40].

A infraestrutura do Programa *Copernicus* contempla três componentes:

- **Componente Espaço** – respeitante à observação da Terra por intermédio de sensores presentes em satélites; constituída por seis famílias de satélites conhecidos como *"Sentinel"* e outras missões contribuintes operadas por entidades nacionais, europeias e internacionais que fornecem dados aos serviços do *Copernicus*.
- **Componente de Terra (In Situ)** – contempla redes de monitorização constituídas por uma variedade de sensores em terra, no mar e no ar cujos dados poderão ser consumidos pelos serviços do *Copernicus* e usados para calibração e validação dos dados obtidos pelos satélites. A gestão destas redes de monitorização é levada a cabo pelos Estados Membros e organismos internacionais.
- **Componente de Serviços** – que visa não só fornecer os dados provenientes das duas componentes anteriores mas também transformar esses dados em informação de valor acrescentado disponibilizada em seis categorias de serviços a entidades interessadas numa política de livre acesso e sem custos.

O programa *Copernicus* conta com investimento público que procura estimular a economia da UE permitindo incentivar áreas como a investigação e o surgimento de empresas prestadoras de serviços com base na informação disponibilizada como, a título de exemplo, empresas que forneçam serviços de avaliação de rendimento de explorações agrícolas recorrendo a imagens multiespectrais fornecidas pelos satélites [41].

Os dados fornecidos por este programa são também de extrema importância para entidades nacionais como o ICNF que, por intermédio da ANPC, utiliza imagens de satélite e ortofotomapas do *Copernicus* para produzir mapas de avaliação de deslizamentos de terra, de risco de erosão e estudos de medidas de mitigação e recuperação de áreas ardidas, entre outros dados de interesse público [42].

2.3.3. Análise e conclusões gerais face ao problema

Foram apresentados na secção anterior alguns exemplos de sistemas de deteção de fogos que podem integrar sistemas de apoio a decisão no auxílio ao combate aos fogos.

Cada uma das soluções expostas apresenta caraterísticas que a tornam adequada para determinados cenários de incêndios florestais e eventualmente desadequada para outros, pelo que é necessário ponderar a sua utilização mediante a situação.

No caso das *Wireless Sensor Networks (WSN)*, estas são uma solução barata para áreas relativamente pequenas podendo dar informação acerca do foco de incêndio e a sua direção de propagação. No entanto, a sua manutenção ainda que possa ser pouco frequente devido à típica autonomia elevada dos nós, seria morosa no caso de redes de grandes dimensões. Além disso, há que também contabilizar a possível perda de nós no caso em que os incêndios consumam completamente a sua área de atuação.

Por outro lado, os SIG, que podem até ser complementados com as *WSN's*, permitem manipular informação geográfica duma forma que auxilia a tomada de decisões, uma vez que permite, por exemplo, conjugar imagens do terreno com *layers* da rede viária e localização de pontos de água contribuindo para que as autoridades responsáveis pelo combate aos incêndios delineiem estratégias de combate suportadas por mais informação.

No entanto, em entrevista ao Comandante Pedro Barreirinha dos Bombeiros Voluntários de Ílhavo [43], foi possível saber que estes sistemas requerem

conhecimento técnico para a sua manipulação, além de que tipicamente estão só disponíveis nos postos de comando fazendo falta no terreno ter acesso direto às suas potencialidades duma forma prática e intuitiva.

No caso particular do sistema MacFire, este tem a particularidade de se poder integrar numa carrinha que pode atuar como um posto de comando móvel no próprio teatro de operações. O seu custo é pouco significativo face a outros sistemas, no entanto a sua utilização carece também da existência de recursos humanos especializados na utilização dos *SIG* e, apesar de ser um posto de comando móvel, poderá estar condicionado pelos acessos existentes no TO.

Já o sistema proposto por Carvalho, F. [47] constituído por uma aplicação *WebSIG*, uma aplicação móvel terrestre e uma aplicação móvel aérea apresenta-se como um sistema muito completo que permite um conjunto de funcionalidades típicas de SIG, visualização de dados, recebimento de alertas mediante certas condições e análise de comportamento de fogo, no entanto apresenta como desvantagem a impossibilidade de operar *offline,* situação que pode ocorrer no TO na eventualidade de perda de cobertura de rede e consequente perda de ligação à Internet.

Além disso, a inclusão de dados meteorológicos das zonas precisas dos fogos permitiria uma simulação da sua propagação mais realista visto que os dados meteorológicos do locais vizinhos poderão não ter em conta a influência do microclima criado pelo fogo na zona da ocorrência.

A utilização de imagens de satélite torna-se também uma solução interessante por fazer uso dos sensores a bordo que permitem auxiliar tanto na fase de deteção mas acima de tudo na fase posterior ao incêndio para fazer o apuramento da área ardida.

Esta abordagem apresenta, no entanto, diversas limitações como é o caso do período de nova passagem tendencialmente longo o que se traduz numa atualização lenta dos dados duma determinada localização, fazendo com que se torne inadequada para situações em que é necessária informação frequente sobre o estado do incêndio.

Além disso também há que ter em conta a reduzida resolução espacial que pode não permitir a deteção de incêndios de pequenas dimensões. O sensor *VIIRS* apresenta uma resolução cerca de três vezes superior ao *MODIS*, no entanto, no caso do sistema descrito por Maden, U. [37] em que as imagens dos satélites só estão disponíveis duas vezes por dia, poderá não ser o suficiente para detetar incêndios numa fase precoce e assim iniciar o combate antes que assuma proporções significativas.

Assim sendo, seria interessante haver um sistema que reunisse as potencialidades das *WSN's* na medida em que houvesse informação rápida sobre o início e a origem do incêndio, assim como a sua propagação, com as potencialidades das ferramentas SIG que permitissem às autoridades responsáveis pela gestão do combate aos incêndios terem uma interface intuitiva onde pudessem decidir rápida e eficazmente quanto às ações a tomar no teatro de operações.

2.4. Plataforma para nó de deteção remota

2.4.1. Particle Electron

A *Particle Electron* é uma plataforma de desenvolvimento criada pela empresa *Particle* para soluções que necessitem de acesso à rede celular (2G/3G). Tem a grande vantagem de trazer um cartão SIM com extensa cobertura de rede e um plano de dados adequado para aplicações de baixa largura de banda, como por exemplo uma rede sensorial, além de ser controlado por um sistema operativo proprietário que permite a rápida e fácil integração do dispositivo com as ferramentas de desenvolvimento e plataforma *Cloud* da *Particle*.

Na Tabela 1 são apresentadas as principais características desta plataforma [8]:

Tabela 1 - Especificações da Particle Electron

Microprocessador Principal	STM32F205 120 MHz ARM Cortex M3
	1 MB de memória Flash
	128 KB de memória RAM
Rede Celular	uBlox SARA U260/U270 (versão 3G)
	uBlox SARA G350 (versão 2G)
	Cartão SIM com cobertura global da Telefonica
Outras especificações	Plano de dados para IoT da Particle
	LED RGB *on-board*
	Bateria LiPo de 2000 mAh
	Sistema Operativo *Real-time* (FreeR-TOS)
	Certificação FCC/CE/IC

Esta plataforma torna-se interessante para aplicações que envolvam a transmissão de dados em ambientes que não disponham de uma infraestrutura de *Wifi* e que a única comunicação de longa distância possível seja pela rede celular.

Um exemplo de aplicação é o apresentado no seguinte *case study* em que a empresa *OptiRTC, Inc*. recorreu ao *Particle Electron* para integrar o seu sistema de gestão de águas pluviais denominado *Continuous Monitoring and Adaptive Control (CMAC)*. Este sistema monitoriza as previsões meteorológicas e com base nelas atua em válvulas de drenagem para minimizar os efeitos das cheias e escoamentos de matérias tóxicas. Até à integração da *Particle Electron*, esta empresa detinha uma infraestrutura IoT muito heterogénea que não correspondia inteiramente às funcionalidades necessárias. A *Particle Electron* e todas as funcionalidades que a plataforma da *Particle* ofereciam, permitiram dar mais controlo a esta

empresa de forma a desenvolver o seu *firmware* e o atualizar via *OTA (Over The Air)* [9].

2.4.2. ESP 32

O *ESP32* é um microcontrolador de baixo custo e de baixo consumo energético desenvolvido pela *Espressif Systems* que conta com um *SoC (System-on-a-chip)* com *Wifi*, *Bluetooth BR/EDR (Classic)* e *Bluetooth 4.2 BLE (Bluetooth Low Energy)*. O *Wifi* no *ESP32* opera em 3 modos distintos: *Soft-AP*, *STA* e *AP+STA*.

Este microcontrolador conta com um microprocessador *Xtensa 32-bit LX6* da *Tensilica* nas versões *single-core* (exclusivamente no modelo *ESP32-S0WD*) e *dual-core* que opera entre os 160 e 240 MHz e realiza até 600 DMIPS. Também dispõe de um co-processador *ULP (Ultra Low Power)* que integra o sub-sistema de baixo consumo e permite, em conjunto com a memória lenta *RTC*, executar um programa em modo *Deep Sleep* que aceda a *timers* internos, dispositivos periféricos ou sensores internos.

Isto é particularmente interessante em aplicações *low power* em que o microcontrolador esteja em modo *deep sleep* e seja acordado por eventos externos ou por um *timer*. Na Figura 2 encontra-se o diagrama de blocos do *ESP32*:

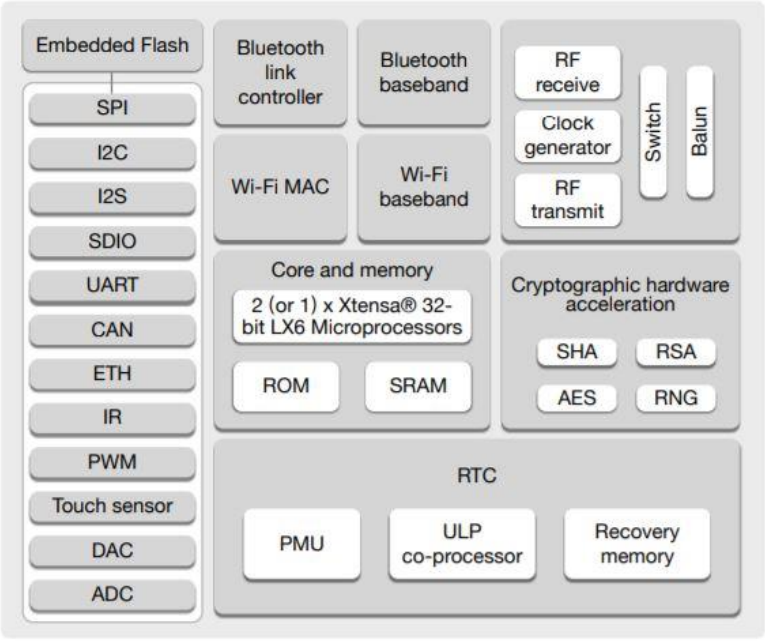

Figura 3 - Diagrama de blocos funcional do ESP32

Conta também com 520 Kb de memória SRAM, 4 Mb de memória *Flash* e 448 Kb de memória *ROM*. Para além disto também põe à disposição uma série de periféricos como 2 8-bit *DAC's*, um *SAR ADC* de 12-bit, 3 interfaces *UART*, entre outros. Um resumo de algumas das principais caraterísticas do *ESP32* encontra-se na Tabela 2 [10] [11]:

Tabela 2 - Especificações do *ESP32*

Características do *ESP32*	**Processadores**	*Xtensa* 32-bit LX6 da *Tensilica* (até 240 MHz)
		ULP (Ultra Low Power co-processor)
	Conectividade	802.11 b/g/b (802.11n @ 2.4 GHz até 150 Mbit/s)
		Bluetooth v4.2 BR/ERD e *Bluetooth Low Energy (BLE)*
	Memória	520 Kb *SRAM*
		4 Mb *Flash*
		448 Kb *ROM*
		16 Kb *SRAM* no *RTC*
	Timer's	*Timer RTC*
		Watchdog RTC
		Oscilador de cristal externo de 32 kHz para o *RTC*
		Oscilador interno de 8 MHz

	Periféricos I/O	*SAR ADC* de 12-bit até 18 canais
		2 *DAC's* de 8-bit
		2 Interfaces I^2S
		2 Interfaces I^2C
		3 Interfaces *UART*
		4 Interfaces *SPI*
		CAN 2.0
		Sensor *HALL*
		10 *GPIO* com suporte a *capacitive touch*
		16 Canais de *PWM*
		34 *GPIO's* programáveis
	Consumo Energético	5 µA em *Hibernation*
		Activo – até 240 mA
		Até 150 µA em *Deep Sleep*

É importante referir que existem atualmente diversas variantes de placas de desenvolvimento *ESP32* e inclusive dos microcontroladores *ESP32* que apresentam versões com diferente número de *GPIO's* e de capacidade de memória ou até número de *cores* do processador diferentes (versões *single-core* e *dual-core*). A título de exemplo temos a placa de desenvolvimento *LoPy 4* da *Pycom* que para além do *SoC* do *ESP32* também apresenta um *transceiver* para *LoRa* e *Sigfox* baseado no *Semtech SX1276* [12]. Todas estas caraterísticas fazem do *ESP32* uma plataforma de *hardware* adequada para aplicações de *IoT*.

Um exemplo de aplicação foi demonstrado por Shatadru Bipasha Biswas e M. Tariq Iqbal em que recorreram às potencialidades desta plataforma para desenvolver um sistema automatizado de bombeamento de água *low-cost* para agricultores de países em desenvolvimento que não tenham os meios financeiros para instalar sistemas baseados em controladores *PLC*. Este sistema permitia, para além do controlo do processo, monitorizar remotamente o sistema através dum *webserver* que corria no próprio *ESP32* [13].

2.4.3. Raspberry Pi 3 B+

O *Raspberry Pi 3 Model B+* é um dos mais recentes *single-board computers* que conta com o processador *BCM2837B0*, *Cortex-A53 64-bit SoC* da *Broadcom* que opera a 1.4 GHz e 1 Gb de memória *LPDDR2 SDRAM*. Na Tabela 3 são apresentadas algumas das caraterísticas mais relevantes desta plataforma [14]:

Tabela 3 - Especificações do *Raspberry Pi 3 B+*

Processador:	*Broadcom BCM2837B0*, *Cortex-A53* 64-bit *SoC* @ 1.4 GHz
Memória:	1GB *LPDDR2 SDRAM*
Conectividade:	*Wifi dual-band* 2.4/5.0 GHz IEEE 802.11.b/g/n/ac *wireless LAN*, *Bluetooth 4.2*, *BLE*
	Gigabit Ethernet através do USB 2.0 (máximo 300 Mbps)
	4 portas USB 2.0
I/O	*Header GPIO* de 40 pinos
Vídeo e Som	1 porta *HDMI* full size
	MIPI DSI display port
	MIPI CSI camera port

Entrada de Energia	5V/2.5ª DC via conector micro USB
	5V DC via header de GPIO
	PoE (*Power over Ethernet*)

Ao contrário das plataformas apresentadas anteriormente que se baseavam em *MCU's*, o *Raspberry Pi 3 B+* é um *single board computer* que apresenta um conjunto de periféricos mais extenso e uma capacidade de processamento superior contando inclusive com uma *GPU* e com interface de entrada e saída de vídeo.

Consequentemente, o seu consumo energético poderá ser significativamente superior sendo que o consumo de corrente típico da plataforma sem periféricos USB ligados é de cerca de 500 mA podendo ser superior com periféricos ligados para um máximo de corrente para a interface USB de 1.2A [15].

Assim sendo esta plataforma seria interessante para aplicações que envolvessem a recolha de imagens ou vídeo, no entanto, para aplicações mais simples de monitorização que envolvam apenas a leitura de alguns sensores, torna-se uma solução dispendiosa tanto a nível de custo do equipamento como a nível de consumo energético, fazendo das outras soluções apresentadas opções mais interessantes.

Um exemplo de um sistema que envolve videovigilância é o sistema desenvolvido por Waheed, S. A., & Sheik Abdul Khader, P. [16] que se apresenta como uma solução significativamente menos dispendiosa do que um sistema de vigilância *CCTV* ao recorrer à plataforma *Raspberry Pi* em conjunto com uma câmera e um algoritmo de *computer vision* recorrendo à *framework SimpleCV* e à linguagem de programação *Python*. Este sistema está especialmente destinado à deteção de queda de pessoas idosas e gera um alerta para os seus/suas cuidadores/as através de *SMS* ou chamada telefónica.

2.4.4. Análise Comparativa das Plataformas

Do ponto de vista aplicacional, as três soluções apresentadas podem ser usadas em cenários diferentes.

A *Particle Electron* é adequada para aplicações que necessitem de acesso à rede celular 2G/3G para a transmissão de dados. Um exemplo de aplicação poderia ser enquanto *gateway* de uma rede de sensores numa zona remota em que não houvesse acesso a uma infraestrutura física de rede.

O *ESP32* é uma solução interessante de baixo consumo que poderá integrar redes de sensores enquanto nó e enquanto *gateway* dado o facto de se poder criar um *web server* que disponibilize dados sensoriais duma rede de sensores.

Finalmente o *Raspberry Pi* seria mais adequado para soluções em que houvesse a necessidade de captação de imagem ou vídeo, como por exemplo, um sistema de vigilância florestal.

Cada solução deverá ser ponderada consoante os requisitos do projeto em que será integrada.

2.5. Comunicações

Nesta secção serão apresentadas algumas tecnologias de transmissão de dados que poderão integrar uma solução de *remote sensing*.

2.5.1. Short-range

2.5.1.1. Wifi

O *Wifi* (*Wireless Fidelity*) é uma tecnologia de transmissão de dados também conhecida por IEEE 802.11 que tem como objetivo permitir estabelecer uma ligação de alta velocidade entre dispositivos. O *standard* IEEE 802.11 estabelece

diretrizes de implementação das camadas *MAC* e *PHY* para as bandas de 2.4, 5 e 60 GHz.

Atualmente os protocolos mais utilizados são o IEEE 802.11b, IEEE 802.11g e IEEE 802.11n que atingem velocidades de transferência máximas de até 11 Mbit/s, 54 Mbit/s e 600 Mbit/s, respetivamente, sendo que estes estabelecem especificações para minimizar a sua suscetibilidade a interferência recorrendo a métodos de transmissão e codificação como o *Direct-Sequence Spread Spectrum (DSSS)* e *Orthogonal Frequency-Division Multiplexing (OFDM)* [18].

Para que haja a transmissão de dados, os dispositivos necessitam de estar conectados de uma de duas formas: a uma *access point* (AP); ou a outros dispositivos que detenham esta tecnologia numa ligação *ad-hoc*. A Figura 4 exemplifica estas duas topologias:

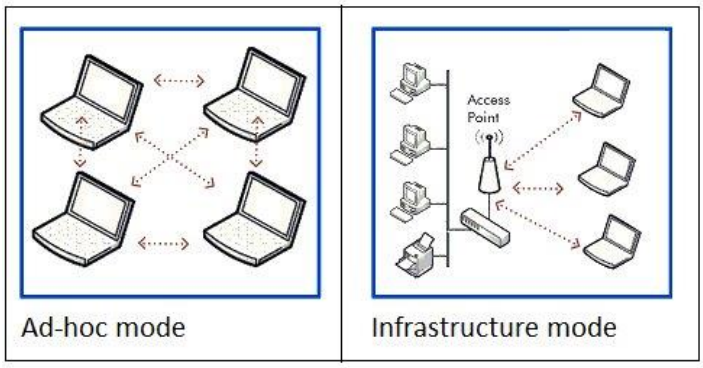

Figura 4 - Modo Ad-hoc e Access Point (AP) [17]

No modo *AP*, existe um dispositivo designado *Access Point* que tipicamente será um *router*, e que mediará o acesso entre dispositivos numa rede para a troca de dados.

Por outro lado, numa ligação do tipo *ad-hoc*, dois dispositivos *Wifi-enabled* poderão comunicar diretamente numa ligação do tipo *point-to-point* sem necessitarem de um dispositivo intermediário. Em ambos os modos, há no entanto a necessidade de que a ligação seja estabelecida, quer diretamente quer mediante um *router*, para que haja a transmissão de dados. [18].

Esta tecnologia opera na mesma banda de frequências que o *Bluetooth* pelo que surge a questão da interferência mútua que poderão causar. Estes efeitos são no entanto mais notados entre o *Wifi* e o *Bluetooth Classic* em comparação com o *BLE* visto que este último não opera de forma contínua.

Para minimizar os efeitos da interferência são utilizadas técnicas como o *spread-spectrum frequency hopping* em que o dispositivo irá "saltar" entre vários canais (várias frequências) dentro da gama de frequências em que opera, de forma a minimizar a interferência. Desta forma mais dispositivos poderão utilizar o espetro [19].

2.5.1.2. Bluetooth BLE

O que é?

Bluetooth BLE é um protocolo de comunicação sem-fios de baixo consumo energético desenhado pela *Bluetooth Special Interest Group (Bluetooth SIG)* para comunicação de curta distância entre dispositivos com especial interesse em aplicações de baixa largura de banda [18]. Este protocolo permite, face à versão anterior denominada *Bluetooth Basic Rate/Enhanced Data Rate* (conhecida por *Bluetooth Classic*), uma poupança significativa a nível de consumo energético assim como uma menor complexidade do design mantendo o mesmo alcance [20].

À semelhança do *Bluetooth BR/EDR*, o *BLE* também opera na banda de frequência de 2.4GHz recorrendo a *FSSK (Frequency-Hopping Spread Spectrum)* que transmite ao longo de 40 canais com espaçamento de 2 MHz (3 canais para fazer *advertising* e os restantes 37 para a troca de dados) de forma a aumentar a robustez face à interferência de outros dispositivos que operem na mesma gama de frequências.

Para além disso também disponibiliza várias *PHY* (*physical layers*) que permitem velocidades de transmissão no intervalo de 125 Kb/s a 2 Mb/s com vários perfis energéticos entre 1 mW e 100 mW.

Apesar de ambas as versões operarem na mesma frequência, estas não são compatíveis. No entanto,em dispositivos com as duas interfaces, pode-se utilizar a mesma antena para ambos os protocolos.

Modelo de Software:

A comunicação entre dispositivos recorrendo a *Bluetooth* faz-se segundo um protocolo específico constituído por diversos elementos.

O *ATT (Attribute Protocol)* é responsável pela comunicação entre dispositivos que assumem o papel de cliente ou servidor.

O *GATT server*, dispositivo que cede a informação, é responsável por manter um conjunto de atributos, que correspondem a estruturas de dados que guardam informação como, por exemplo, medições de temperatura, frequência cardíaca, entre outros, e que são geridos pelo *GATT (Generic Attributes)* [21].

Cada atributo é identificado por um identificador universal único denominado *UUID (Universally Unique Identifier)* que corresponde a um número num formato *standardizado* de 128-bit. [22]

O *GATT* define uma estrutura de dados hierárquica que é exposta aos dispositivos *BLE* conectados e que faz uso do *ATT* para efetuar algumas operações como descobrir *UUID's* de *Services*, *Services* para um determinado *UUID*, *Characteristics* para um determinado *Service*, *Characteristics* para um determinado *UUID* e ler *Descriptors* para uma determinada *Characteristic*, sendo que envia respostas aos pedidos assincronamente para o cliente [23].

Estas operações são feitas tendo por base o chamado *GATT Profile* que descreve um uso específico e comportamentos de uma funcionalidade do *GATT*. Este apresenta uma arquitetura que é definida pela Figura 5.

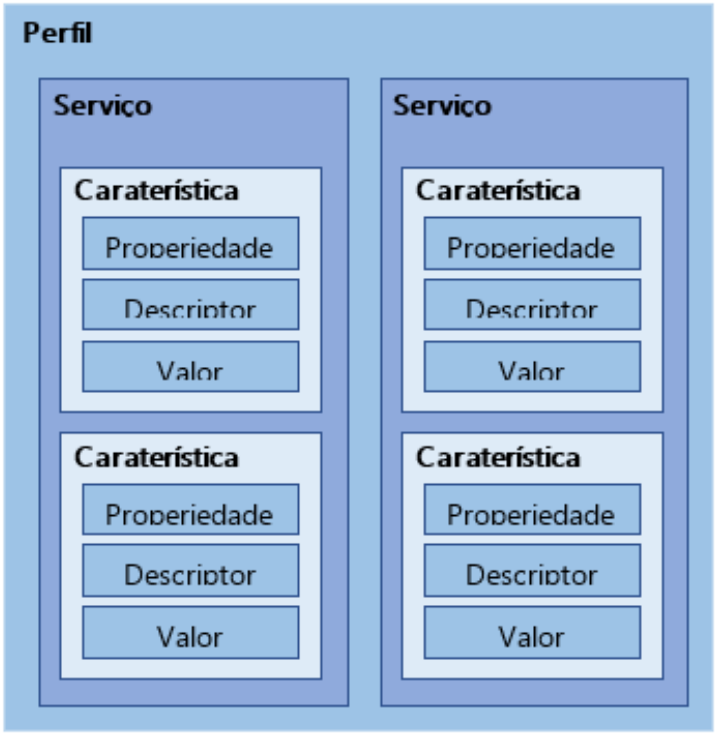

Figura 5 - Estrutura hierárquica de um GATT Profile

A componente de mais alto nível do *GATT Profile* é o *Profile*. O *Profile* é responsável por definir o comportamento do *GATT Client* e do *GATT Server* no que toca aos requisitos dos *Services, Characteristics,* conexões e de segurança.

Este é também um conjunto de *Services* que corresponde a uma conjunto de *Characteristics* que encapsulam uma funcionalidade do dispositivo.

Por sua vez, as *Characteristics* contêm um valor único e um conjunto de propriedades designadas *Descriptors*. Os *Descriptors* são atributos definidos que descrevem uma caraterística em particular [22].

O *GATT* é então responsável por definir o formato dos *Services* e das suas *Characteristics* além de disponibilizar um conjunto de operações para fazer interface com os atributos como a descoberta de *Services,* operações de *Read* e *Write* sobre as *Characteristics,* notificações e indicações [23].

Aplicações do *BLE*:

Ao permitir a implementação de diversas topologias de rede, o *BLE* torna-se adequado para um conjunto mais alargado de aplicações.

Tabela 4 - Topologias de Rede do BLE

Topologia	Tipo de Conexão	Descrição e Aplicações
Point-to-Point	*One-to-one (1:1)*	Permite a transferência de dados entre dois dispositivos. Adequado para aplicações que envolvam, por exemplo, *fitness trackers*, periféricos de computadores, entre outros.
Broadcast	*One-to-many (1:m)*	Otimizada no *BLE* para *Services* de navegação *indoor*, partilha de informação localizada de produtos assim como monitorização de bens.
Mesh	*Many-to-many (m:m)*	Com o *BLE*, torna-se possível a criação de redes de grandes dimensões de dispositivos que partilham os dados entre si [24] permitindo criar diversas rotas para o encaminhamento de informação tornando assim a rede mais robusta. Especialmente adequada para aplicações de automação, controlo e monitorização.

2.5.2. Long-range

2.5.2.1. LoRa

LoRa (que significa *"Long Range"*) é uma tecnologia de comunicação sem fios de longa distância do tipo *Low-Power Wide-Area Network* (LPWAN) especialmente adaptada para aplicações que necessitem de baixo consumo energético, longo alcance e baixa taxa de transmissão de dados.

Com um alcance de cerca de 10-40 km em ambientes rurais e de 1-5 km em ambiente urbano, *LoRa* torna-se uma tecnologia candidata a aplicações de *remote sensing* que necessitem de pouco tráfego de dados e de reduzida manutenção (com um tempo de vida da bateria até 10 anos) [25].

Esta tecnologia subdivide-se em duas camadas: a *LoRa Physical Layer* e a *LoRaWAN network protocol*.

LoRa Physical Layer

A *LoRa Physical Layer*, desenvolvida pela *Semtech*, recorre à técnica de modulação *Chirp Spread Spectrum (CSS)* que utiliza *chirps* com uma variação linear na frequência ao longo do tempo para codificar informação. Devido a esta linearidade os *offsets* de frequência entre o transmissor e o receptor são equivalentes a *offsets* temporais que são facilmente compensados no recetor.

Esta técnica de modulação é também caraterizada por espalhar um sinal *narrow-band* sobre um canal de maior largura de banda, o que tem como consequência uma utilização menos eficiente do espectro. Todas estas caraterísticas conferem a esta tecnologia uma elevada imunidade ao ruído, ao efeitos de *Doppler* e dificultam a sua deteção [25].

Utiliza bandas *ISM* abertas sendo que na Europa utiliza a banda de 868 MHz, na América do Norte a banda de 915 Mhz e na Ásia a banda de 433 MHz com uma taxa de transmissão de dados pode chegar aos 50 Kbps [26].

Estão disponíveis vários parâmetros que permitem configurar a modulação da LoRa face às necessidades da aplicação, são eles:

- ***Spreading Factor (SF)*** – o *LoRa* envia sinais "*chirp*" que a cada incremento do *Spreading Factor* demoram o dobro do tempo a serem transmitidos. Valores elevados de *Spreading Factor* conferem uma maior resistência ao ruído e maior fiabilidade na ligação a custo de um maior *time on air* que se traduz num maior consumo energético e numa taxa de transmissão mais baixa. Este parâmetro está no intervalo de 7 a 12. Cada incremento no *Spreading Factor* corresponde a um incremento de potência de 2.5dB.

- ***Bandwidth (BD)*** – o sinal "*chirp*" ocupa a largura de banda definida por este parâmetro. Uma maior largura de banda traduz-se numa maior taxa de transmissão podendo no entanto traduzir-se numa maior congestão do canal. São disponibilizadas 3 configurações para este parâmetro: 125 KHz, 250 KHz e 500 KHz. Cada nível maior de largura de banda traduz-se numa perda de potência de 3dB.
- ***Coding Rate (CR)*** – este protocolo utiliza técnicas de correção de erros que poderão ser necessárias em canais com muita interferência. São disponibilizados 5 níveis de *CR*, no intervalo de 0 a 4 sendo *CR* = 0 sem correção de erros e sendo os *coding rates* 4/5, 2/3, 4/7 e 1/2 correspondentes aos restantes níveis de 1 a 4, respetivamente. Com o incremento do *Coding Rate* aumenta-se a fiabilidade da ligação mas diminui-se a taxa de transmissão.

LoRaWAN network protocol

LoRaWAN é um protocolo do tipo *Low-Power Wide-Area Network (LPWAN)* que tem por base a tecnologia *LoRa* à qual lhe complementa um mecanismo de *Media Access Control (MAC)* para permitir conectar *end-devices* com uma *gateway* utilizando a modulação desta tecnologia.

A Figura 6 mostra a arquitetura da *LoRaWAN*:

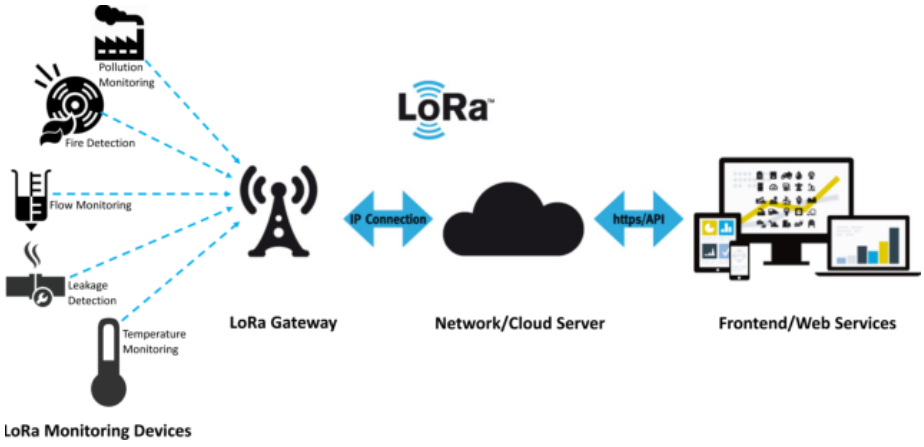

Figura 6 - Arquitetura da *LoRaWAN* [27]

A arquitetura apresentada tem o seguinte funcionamente: os *end-devices* (dispositivos que disponham da interface de comunicação *LoRa*) comunicam com as *gateways* recorrendo a *LoRa*, que por sua vez reencaminham os dados para um servidor através duma interface de maior velocidade, geralmente *Ethernet* ou 3G/4G.

O servidor poderá eventualmente expor uma *API* (*Application Programming Interface*) que permita a aplicações consumirem os seus dados. É de salientar que esta arquitetura é bidirecional pelo que os *end-devices* tanto poderão ser, a título de exemplo, sensores que comunicam para o servidor os seus dados, como também poderão ser atuadores, como uma electroválvula, que recebe comandos provenientes do servidor.

Assim as *gateways* são apenas encaminhadores de pacotes que fazem a interface entre uma conexão *LoRa* e uma conexão *IP*.

2.5.2.2. NB-IoT

NB-IoT é, à semelhança do *LoRa*, uma tecnologia de comunicação sem fios de longa distância do tipo *Low-Power Wide-Area Network* (*LPWAN*) proposta pela *3GPP* que faz uso da infraestrutura da rede *LTE*. Esta tecnologia pode ser implementada em três modos distintos [28]:

- ***Stand alone operation*** – nesta hipótese o *NB-IoT* utiliza um canal *GSM* (200 kHz) sobrando ainda um intervalo de guarda de 10 kHz em ambos os lados do espetro, utilizando uma portadora de 180 kHz.
- ***Guard band operation*** – deste modo o *NB-IoT* utiliza o intervalo de guarda de uma portadora *LTE* em que recorre aos blocos de recursos não utilizados na banda de guarda da portadora.
- ***In-band operation*** – nesta implementação o *NB-IoT* utiliza os blocos de recursos no interior da portadora *LTE*.

Na Figura 7 apresenta-se uma representação dos três modos distintos:

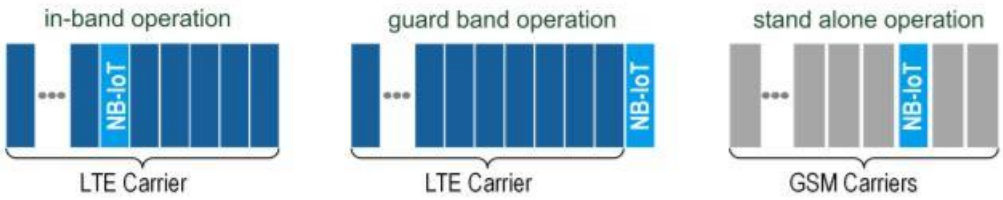

Figura 7 - Os três modos de implementação do *NB-IoT* [28]

Este protocolo disponibiliza um modo de multiacesso utilizando a tecnologia *Orthogonal Frequency-Division Multiple Access* (OFDMA) em que é atribuída uma sub-portadora de 15 kHz a cada dispositivo no *uplink*

Além disso, as transmissões no *uplink* podem ser feitas em conjunto numa menor largura de banda ao eliminar o espaçamento da sub-portadora de 3.75 kHz [26] [29]. O *NB-IoT* apresenta uma taxa de transmissão de dados de 160-200 kHz no *uplink* e de 160-250 kHz no *downlink*, contando com uma cobertura de 18 km em ambiente urbano e de 25 km em ambiente rural.

Ao haver uma coexistência do *LTE* com o *NB-IoT,* poderá haver interferência no *LTE* celular. Um exemplo de um estudo do efeito desta coexistência foi realizado por Oh, J., & Song, em [30] onde mostraram que a interferência causada pelo LTE utilizando o espaçamento de 3.75 kHz da portadora de *uplink* do *NB-IoT* ocorria no espaçamento da portadora de *uplink* de 15 kHz do *NB-IoT*.

Os autores verificaram que isto acontecia pois o modo *in-band* do *NB-IoT* não conseguia alocar recursos do *LTE*, interferindo com o *LTE* em ambos os lados da banda alocada ao *NB-IoT*.

De forma a reduzir a interferência, uma das sugestões dos autores é fazer com que o *NB-IoT* iguale a potência por portadora à do *LTE*.

3

Trabalho desenvolvido

Neste capítulo será apresentado o trabalho desenvolvido no projeto desta dissertação. Começam por ser identificados os requisitos do sistema por um grau de importância conferido, que visam responder aos objetivos delineados. Segue-se a apresentação dos requisitos de *hardware* e *software* do *device* e da aplicação *tablet* que deverão ter para implementarem os requisitos do sistema propostos. Posteriormente é feita a análise da implementação do *device* e da aplicação *tablet* onde começam por ser explicadas as opções tecnológicas tanto a nível da plataforma de *hardware* como das ferramentas utilizadas na aplicação e apresentadas e discutidas as arquiteturas pensadas para ambas as componentes do sistema. É também explicada a arquitetura do *Feature* Service desenvolvido no *ArcGIS Online*. Todas as fases de implementação do projeto nas três componentes serão explicadas pormenorizadamente ao longo deste capítulo.

3.1. Requisitos do Sistema

Tendo por base os objetivos delineados para o projeto desta dissertação, que tiveram em conta o que foi discutido nas reuniões do projeto foRESTER e que contaram também com o contributo de reuniões particulares com entidades potencialmente beneficiárias do projeto além das sugestões do professor orientador, estabeleceu-se um conjunto de requisitos a que o sistema deve cumprir segundo uma escala de importância que procurava priveligiar as funcionalidades apontadas como prioritárias. A Tabela 5 apresenta os requisitos previstos para o sistema.

Tabela 5 – Requisitos do Sistema

Requisitos	Descrição	Importância
Requisito 1	Disponibilização dos dados sensoriais do *ESP 32* através do *BLE*	Elevada
Requisito 2	Conexão com da *app* com o *ESP 32* e leitura dos dados sensoriais	Elevada
Requisito 3	Criação de *Feature Layers* no *ArcGIS Online* e de uma *Feature Service* que exponha as *features* e as tabelas com os respetivos dados	Elevada
Requisito 4	Criação de uma *Geodatabase* que sincronize com o *Feature Service* e permita trabalhar *offline*	Elevada
Requisito 5	Implementação de leitura e edição dos atributos (*fields*) de cada *feature* de qualquer tipo (*Point*, *Polygon* ou *Polyline*) e possibilidade de movê-las e apagá-las	Elevada
Requisito 6	Implementação da possibilidade de abrir/adicionar/remover anexos de imagens das *features*	Elevada

	provenientes da câmera ou do armazenamento local	
Requisito 7	Atualização dos atributos e da posição da *feature* respeitante ao *device* aquando da ligação ser estabelecida com este	Elevada
Requisito 8	Medição de comprimento e área de *features* do tipo *Polyline* e *Polygon*, respetivamente	Média
Requisito 9	Sobreposição de *timestamp*, localização e informação relativa ao fogo na foto	Média
Requisito 10	Estimação da distância à frente de fogo e sua georreferenciação	Baixa
Requisito 11	Criação de um cliente em *Python* para consumo dos dados da *Feature Service*	Baixa
Requisito 12	Criação de um *Tile Package* no *ArcGIS Pro* que será carregado localmente como mapa-base (e assim evitar recorrer aos mapas base da *ESRI* disponíveis *online*)	Baixa
Requisito 13	Implementação do acesso ao *Portal ArcGIS* e possibilidade de guardar um mapa desenvolvido localmente	Baixa
Requisito 14	Implementação do *on-demand workflow* que permite descarregar uma área pré-definida de um mapa	Média
Requisito 15	A aplicação deve correr *offline* permitindo a sincronização de dados aquando do restabelecimento de conexão de rede	Alta
Requisito 16	A aplicação deve estar preparada para interpretar qualquer estrutura de dados proveniente do *Feature Service* a que se conectar.	Alta

O projeto desta dissertação está dividido em três partes:

- Concepção do *device*
- Concepção da Aplicação *tablet*.
- Concepção das *Feature Layers* no *ArcGIS Online*

O *device* servirá como uma pequena estação meteorológica local que transmitirá os dados dos parâmetros atmosféricos para a aplicação *tablet* através da interface de comunicação *Bluetooth BLE* ou *Wifi*.

A aplicação consumirá estes dados e exibirá a localização do dispositivo no mapa com os respetivos parâmetros atmosféricos permitindo fazer operações sobre o ponto como: editar, remover, mover e ver/adicionar/remover anexos.

As *Feature Layers* desenvolvidas no *ArcGIS Online* poderão conter informação operacional importante que pode ser exibida e manipulada na aplicação *tablet.*

3.2. Requisitos de *hardware* e *software*

A implementação dos requisitos atrás apresentados carece de um suporte de *hardware* e *software* tanto a nível do *device* como do *tablet* que impõe algumas restrições nos equipamentos utilizados. A Tabela 6 e a Tabela 7 apresentam os requisitos do *device* e do tablet, respetivamente.

Tabela 6 - Requisitos do *device*

Requisitos	Descrição
Requisito 1	Interfaces de comunicação como *UART/I²C/SPI/etc*. para conectar os sensores

Requisito 2	Interfaces de comunicação sem-fios *Bluetooth BLE* e *Wifi*
Requisito 3	Baixo consumo energético
Requisito 4	Configuração fácil e rápida

Tabela 7 - Requisitos do *tablet*

Requisitos	Descrição
Requisito 1	Versão do *Android* mínima 6.0 *Marshmallow (API 23)*
Requisito 2	Conexão de rede como *Wifi* e/ou *GSM/3G/4G*
Requisito 3	Possuir *GPS*

3.3. Implementação do *device*

A implementação do *device* inicia-se com a discriminação das funcionalidades que este deve possuir. O propósito deste equipamento será o de fazer a leitura dos dados de alguns sensores e transmitir sobre a forma de uma estrutura de dados pré-definida pelas interfaces de comunicação sem-fios existentes, tal como é elucidativo na Figura 8.

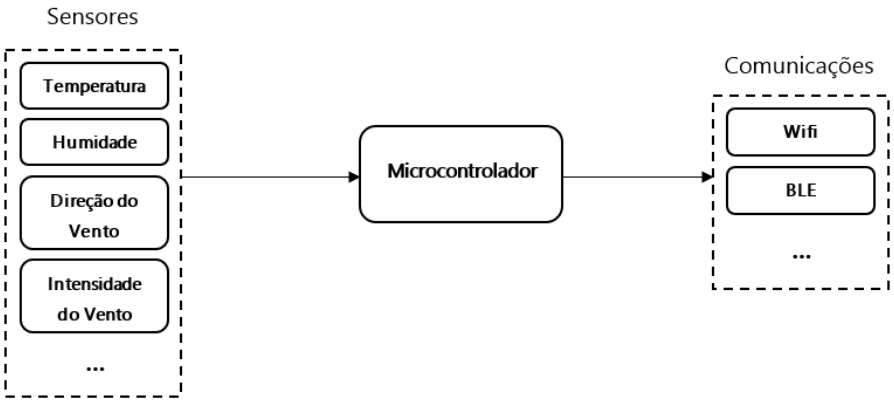

Figura 8 - Esquema geral de funcionamento do *device*

A existência de duas interfaces de comunicação sem-fios permite conferir alguma redundância na comunicação de forma a haver uma alternativa em caso de indisponibilidade de uma delas. A interface de comunicação priveligiada será a do *Bluetooth BLE* dado que oportunamente a aplicação necessitará de ligação de Internet para sincronizar as alterações no mapa e assim evita-se a desconexão com o *device* para sincronizar com o *Feature Service*. As principais caraterísticas do *device* serão as seguintes:

- Leitura periódica de dados sensoriais através de interfaces como *I²C/UART/etc*.
- Disponibilização dos dados sensoriais numa estrutura de dados uniformizada através do *Bluetooth BLE* ou numa página *HTML* através do *Wifi*.
- Possibilidade de conexão em ambas as interfaces num regime mutuamente exclusivo (por limitações do *hardware* escolhido).

O *device* é constituído por um *MCU*, sensores atmosféricos que irão medir parâmetros como temperatura, humidade e concentrações de gases como o CO e CO_2, e um recetor *GPS*. O *MCU* utilizado é o *ESP32* dado que este já apresenta como opções de conetividade *Wifi* e *Bluetooth BLE*. São disponibilizadas ambas as interfaces de comunicação sendo que estas são mutualmente exclusivas tendo em conta as limitações físicas do ESP32 que apresenta somente uma antena para ambas as interfaces.

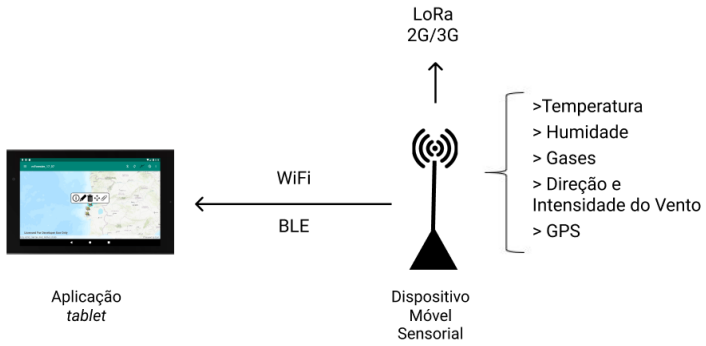

Figura 9 - Esquema de comunicação entre a aplicação e o *device*

Para o *Wifi* foi criado um *webserver* que expõe os dados sensoriais, localização e identificador do dispositivo no formato *JSON* e para o *BLE* foram criados um *Service* para os dados atmosféricos (denominado *Weather*), outro para a localização (denominado *Location*) e finalmente outro para leitura e escrita do *ID* do dispositivo de forma a identificá-lo no mapa (*Device Service*). Em cada *Service* foram criadas várias *Characteristics* respeitantes a cada parâmetro de interesse.

Os sensores utilizados são um sensor de temperatura e humidade e um receptor de *GPS*. O sistema está pensado para ser escalável sendo possível adicionar ao *ESP32* mais sensores mediante a sua compatibilidade com as interfaces de comunicação e criar as *Characteristics* para cada parâmetro sensorial para além de ser também fácil ler mais *Characteristics* na aplicação sendo apenas necessário ativar as notificações para elas.

3.3.1. Opções tecnológicas do *device*

A escolha da plataforma de *hardware* a utilizar teve por base a existência de duas interfaces de comunicação, preferencialmente *Wifi* e *Bluetooth BLE* por estarem presentes na maioria dos *smartphones* e *tablets*. Sendo duas interfaces conferia-se assim alguma redundância nas comunicações tornando o *device* mais robusto. Tendo por base o objetivo do projeto desta dissertação de construção de um equipamento recorrendo a plataformas de *hardware* de baixo custo partiu-se de um conjunto de soluções que cumpriam este requisito.

A primeira hipótese equacionada foi a utilização da plataforma *Particle Electron* em conjunto com dois módulos de *Bluetooth BLE* (como o *HM-10*) e de *Wifi* (como o *ESP-01*), cujo esquema de ligação se encontra representado na Figura 10.

No caso desta solução, ambos os módulos utilizam a interface *UART* para comunicação. Dado que esta plataforma dispõe de três interfaces *UART*, utilizam-se assim duas destas interfaces para se ligarem estes módulos.

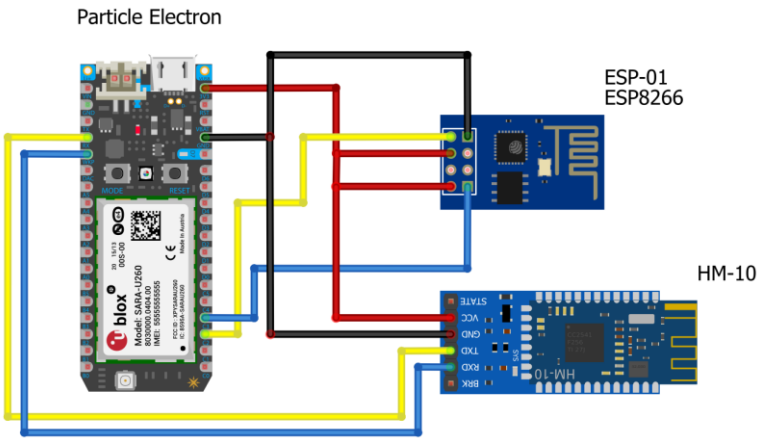

Figura 10 - Esquema de ligação de um *Particle Electron* com os módulos ESP-01 (Wifi) e HM-10 (BLE)

Esta solução, no entanto, não foi considerada dado que apresenta um *footprint* elevado, tem um custo um pouco mais significativo e existem já alternativas que agregam todas as potencialidades desta solução num só *chip*.

A segunda hipótese estudada foi o *Raspberry Pi* que também dispõe de *Wifi* e *BLE*, no entanto esta solução não é um microcontrolador mas sim um *single board computer* que tem uma capacidade de processamento significativamente superior à custa de um consumo energético substancialmente mais elevado. É por isso uma solução que seria considerada em caso de necessidade de aquisição de imagem ou vídeo mas que para o objetivo do projeto, que não requer elevada capacidade de processamento, torna-se cara e desaquada face a outras soluções ponderadas.

Finalmente, a hipótese que acabou por ser escolhida e que dispunha das interfaces de comunicação desejadas, foi a plataforma *ESP32*. Esta plataforma tem a vantagem de ser de baixo custo, de dispor num só *chip* de *Wifi* e *BLE* e além disso de ter uma boa capacidade de processamento face às necessidades.

Tem também a vantagem de ser facilmente programada através do ambiente de desenvolvimento do Arduino e de ter muita documentação disponível.

3.3.2. Arquitetura do *device*

3.3.2.1. Esquema de montagem do *device*

Escolhida a plataforma para desenvolvimento do *device* passou-se então à elaboração da sua arquitetura. Em primeiro lugar foram escolhidos os sensores a serem incluídos que foram um sensor de temperatura e humidade e um receptor de *GNSS*. O sensor de temperatura e humidade escolhido foi o *HTU21D* com uma precisão de humidade relativa de ±2% @25ºC e uma precisão de temperatura de ±0.3ºC @25ºC que comunica através da interface I^2C.

Já o receptor de *GNSS* que se optou foi o *NEO-M8N* de 72 canais e de sensibilidade até -167 dBm que comunica através de interface *UART*. O esquema de ligação da montagem efetuada encontra-se na Figura 11.

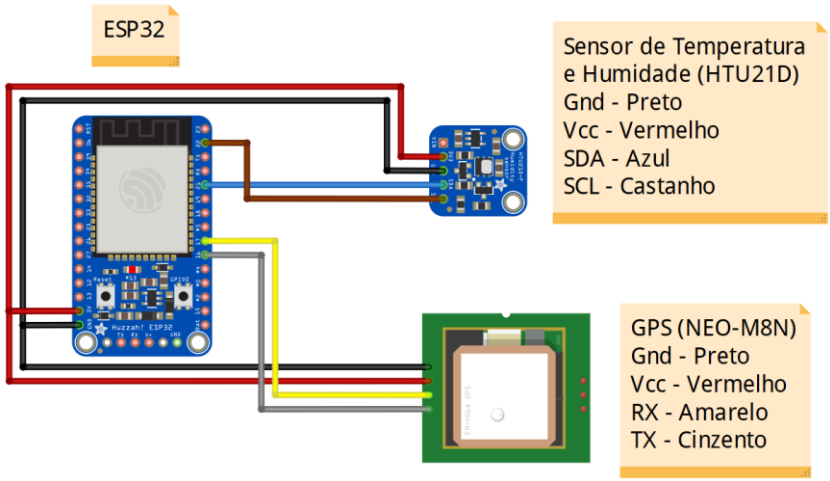

Figura 11 - Esquema de ligação do *device*

3.3.2.2. Estrutura de dados

Após a escolha dos sensores a utilizar e feita a sua montagem, inicia-se o planeamento da estrutura de dados a adoptar para ambas as interfaces de comunicação do *ESP32* utilizadas.

- *BLE*

A estrutura de dados utilizada no *BLE* tem em conta as caraterísticas do protocolo subjacente. Criaram-se três *Services* designados "*Location*", "*Weather*" e "*Device*".

O *Service* "*Location*" contém uma *Characteristic* que detém a latitude e longitude no formato "*lat;lng*". As coordenadas são depois retiradas na aplicação evitando-se assim criar duas *Characteristics*.

O *Service* "*Weather*" contém uma *Characteristic* com os dados de temperatura e humidade no formato "*temp;hum*", pelas mesmas razões que o anterior.

Finalmente o *Service* "*Device*" contém duas *Characteristics,* uma para receber o *ID* (identificador) do *device* e outra para configurar esse mesmo identificador no dispositivo.

Na Figura 12 encontra-se representada a estrutura de dados do *BLE*.

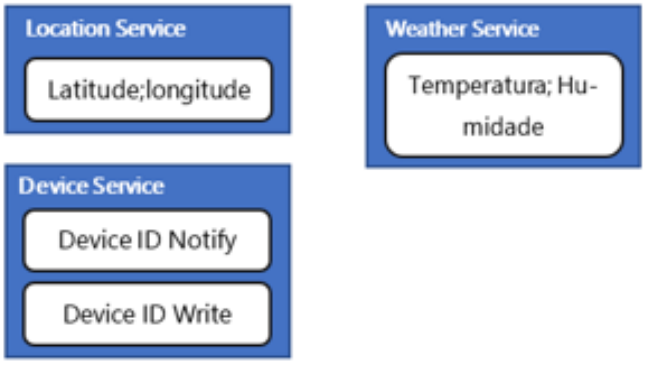

Figura 12 - Estrutura de dados dos *Services Weather*, *Location* e *Device ID*

Tanto os três *Services* como as suas *Characteristics* têm um identificador único designado *Universal Unique ID (UUID)* que consiste num número único de 128-bit usado para identificar *Services, Characteristics* e *Descriptors* e que foi gerado individualmente através de uma ferramenta *online*.. Existem *UUIDs* predefinidos pelo Bluetooth SIG de 16-bit que podem ser utilizados na condição de se cumprir a especificação a que estão atribuídos no entanto, de forma a se poder

criar *Services* e *Characteristics* personalizados, é necessário utilizar *UUIDs* que não estejam atribuídos.

Na Figura 13 encontra-se um excerto do código para configurar o *BLE* no *device*.

```
/* ********************************************************************************************
   initBLE() -> method to initialize the ESP32's BLE interface
*/
void initBLE() {
  // Create the BLE Device
  BLEDevice::init("mForrester-BLE"); // device's name

  // Create the BLE Server and set its callback
  bleServer = BLEDevice::createServer();
  bleServer->setCallbacks(new ServerCallbacks());

  // Create the BLE Services
  BLEService *weatherService = bleServer->createService(WEATHER_SERVICE_UUID);
  BLEService *locationService = bleServer->createService(LOCATION_SERVICE_UUID);
  BLEService *idService = bleServer->createService(ID_SERVICE_UUID);

  // Create BLE Characteristics
  charTempHum = weatherService->createCharacteristic(CHARACTERISTIC_TEMPHUM_UUID,
                                        BLECharacteristic::PROPERTY_NOTIFY);
  charLatLng = locationService->createCharacteristic(CHARACTERISTIC_LATLNG_UUID,
                                        BLECharacteristic::PROPERTY_NOTIFY );
  charDeviceIdNotify = idService->createCharacteristic(CHARACTERISTIC_DEVICEID_READ_UUID,
                                        BLECharacteristic::PROPERTY_NOTIFY );
  charDeviceIdWrite = idService->createCharacteristic(CHARACTERISTIC_DEVICEID_WRITE_UUID,
                                        BLECharacteristic::PROPERTY_WRITE );

  // Add Descriptors
  charTempHum->addDescriptor(new BLE2902());
  charLatLng->addDescriptor(new BLE2902());
  charDeviceIdNotify->addDescriptor(new BLE2902());

  // Set Callbacks
  charDeviceIdWrite->setCallbacks(new CharacteristicCallbacks());

  // Start the service
  weatherService->start();
  locationService->start();
  idService->start();

  // Start advertising
  bleServer->getAdvertising()->start();

  Serial.println(F("BLE interface initialized!"));
}
```

Figura 13 - Excerto de código para configuração do BLE no device

Em primeiro lugar começa-se por configurar o *device* atribuindo-lhe um nome. Este *device* comportar-se-à como o *GATT Server* que fornece dados dos sensores ao ESP32. É também definida uma função *callback* que será chamada quando haja eventos de conexão ou desconexão de *GATT Clients* com o *device*.

Posteriormente são criados os três *Services* que foram definidos na arquitetura e as *Characteristics* que os pertencem. A cada um dos *Services* e das *Characteristics* é atribuído um identificador único (UUID) de 128-bit gerado por um ferramenta *online*.

Na criação das *Characteristics* existe um segundo parâmetro que define as suas propriedades que no caso do *device* serão de notificação (*Notify)* para as *Characteristics* de latitude/longitude, temperatura/humidade e leitura do *ID* do dispositivo, e de escrita (*Write*) para a configuração do *ID* do dispositivo. A propriedade de notificação permite informar periodicamente o *GATT Client* de novos valores a partir do momento em que este subscreve a essas alterações. Desta forma evitam-se sucessivos pedidos de leitura (*Read*) ao *GATT Server* que consumirão recursos computacionais na parte do *GATT Client*.

De seguida, é adicionado o *Descriptor* de *UUID* "0x2902" designado "*Client Characteristic Configuration*" que permite ao *GATT Client* ativar as notificações no *GATT Server* para que este possa ficar preparado para transmitir os dados dos sensores.

Depois é definida uma função de *callback* para a *Characteristic* de escrita do *ID* do dispositivo que contém a função *onWrite(BLECharacteristic *characteristic)* que será chamada quando for atribuído um novo *ID* ao *device* para o guardar em memória.

Finalmente são iniciados os *Services* pré-definidos com as suas *Characteristics* e o *GATT Server* disponibiliza-os para consumo pelos *GATT Clients*.

- *Wifi*

No que toca à estrutura de dados utlizada no *Wifi,* optou-se por fornecer os dados sensoriais no formato *JSON* através duma página *HTML*.

Quando um cliente *Wifi* estabelece uma conexão com o *device* este faz a leitura dos sensores e constrói um objeto *JSON*. Esse objeto é depois inserido na *tag <body>* da página *HTML* para ser disponibilizado no *webserver*. Na *tag <head>* é inserida uma *meta-tag* que atualiza a página a cada três segundos e assim permitir ao cliente aceder a dados sensoriais atualizados.

Na Figura 14 encontra-se representada a estrutura de dados da mensagem no formato *JSON* disponibilizada pelo *device*.

```
{
   "weather": {
      "temp;hum" : "",
      ...
   },
   "location": {
      "lat;lng" : "",
   },
   "device": {
      "deviceIdNotify" : "",
      "deviceIdWrite" : "",
   }
}
```

Figura 14 - Dados do device no formato JSON

O *device* está configurado no modo *Access Point (AP)* para que possa operar como um *webserver* e assim fornecer uma página *HTML* com os dados sensoriais no formato *JSON*. Desta forma não é necessário haver um *router* intermediário entre o cliente *Wifi* e o *device* podendo estes se conectarem diretamente numa ligação do tipo *point-to-point*.

A configuração do *Wifi* é realizada, à semelhança da configuração do *BLE*, na função de *setup()* sendo que o *device* ficará posteriormente à espera de conexões iniciadas pelo cliente.

Na Figura 15 encontra-se um excerto do código para configurar o *Wifi* no *device*.

```
/* ********************************************************************
   initSoftAp() -> Configure the ESP32 Wifi in softAP mode
*/
void initSoftAp() {
  // Connect to Wi-Fi network with SSID and password
  Serial.print(F("Setting AP (Access Point)…"));
  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.print(F("AP IP address: "));
  Serial.println(IP);

  server.begin();

  Serial.println(F("ESP32 configured in softAP mode"));
}
```

Figura 15 - Excerto de código para configuração do *Wifi* no *device*

Começa-se por configurar o *device* como *SoftAP* (que significa "*software enabled access point*") passando como parâmetro o nome que se vai atribuir à rede (designado de *SSID*) e a password para o acesso. De seguida obtém-se o endereço *IP* da rede e mostra-se na consola do *Arduino IDE*.

Finalmente é iniciado o *webserver* preparando assim o *device* para receber pedidos de conexão.

3.4. Implementação da *app tablet*

A aplicação, que é destinada para *tablet Android,* tem uma interface de utilizador que contém um mapa que indica o posicionamento dos *devices*, uma janela do tipo *Callout* que surge com a seleção de uma *feature* no mapa e que

contém um conjunto de opções de menu, uma aba do tipo *Navigation Drawer* com a opção de seleção do mapa-base e das *layers* a sobrepor e um menu na *Action Bar* com outras funcionalidades disponíveis.

Na Figura 16 está representado o *layout* da aplicação.

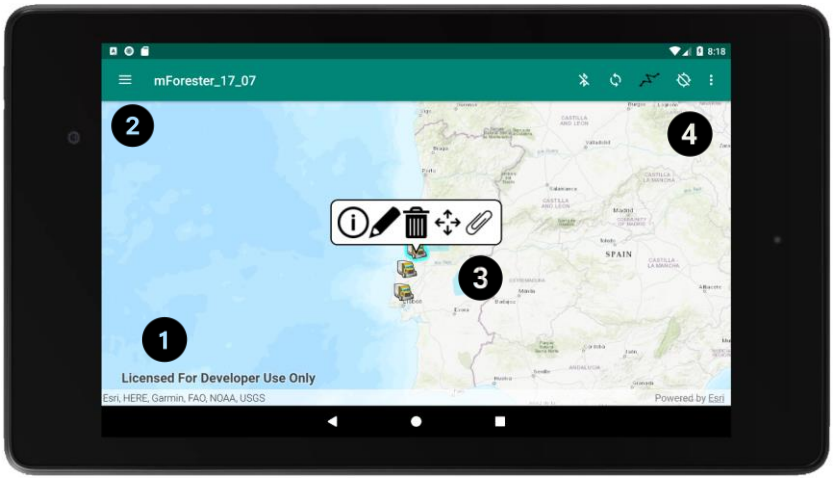

Figura 16 - *Layout* da aplicação *tablet*

1. Área do mapa – nesta zona encontra-se o mapa que terá as *layers* ativas e o mapa-base escolhido.
2. Menu de seleção de *layers* e mapa-base – menu que possibilita a seleção do mapa-base entre uma lista de quatro mapas (topográfico, oceânico, estradas e imagem de satélite) e das *layers* ativas.
3. Menu de opções da *feature* – menu com opções disponíveis sobre as *features* de informação, editar atributos, remover a *feature*, mover a *feature* e aceder aos seus anexos.
4. Menu de funcionalidades – menu com opções de conetividade e outras funcionalidades da aplicação.

A sua implementação começa com a discriminação das funcionalidades desejadas que foram previamente apresentadas nos requisitos. De forma a cumprir

esses requisitos é necessário tomar algumas opções relativas ao desenvolvimento da *app tablet*.

- Sistema operativo para que será desenvolvido (*Android*, *iOS* ou *cross-platform*).
- Linguagem de programação a ser usada.
- Ferramentas, como *SDK's*, *API's*, etc. usadas para o desenvolvimento da *app tablet*.

Após terem sido feitas as escolhas relativamente ao desenvolvimento da *app tablet*, é necessário planear a arquitetura da *app* com os componentes desejados para atingir os requisitos que se apresentaram. Para tal é conveniente ter uma visão modular do sistema em que cada componente interage com outros de forma a cumprir o objetivo final.

3.4.1. Opções tecnológicas da *app tablet*

É então agora necessário efetuar as escolhas relativas ao desenvolvimento da *app tablet* detalhando as razões que levaram à sua opção.

Em primeiro lugar é preciso decidir para que sistema operativo se vai desenvolver a aplicação. As possibilidades consideradas são as de desenvolver nativamente para *Android*, para *iOS* ou optar por uma solução híbrida que compile para as duas plataformas. A solução adotada foi a de desenvolver nativamente para *Android* tendo em conta não só o tempo disponível para a execução da dissertação, mas também porque a escolha do sistema operativo iOS implicava um maior tempo de desenvolvimento dada a curva de aprendizagem inerente a esta opção. É de salientar que plataforma que veio a ser escolhida para a componente do SIG tem um kit de desenvolvimento de software (*SDK*) direcionado para ambas as plataformas *Android*. e iOS.

O segundo passo é escolher a linguagem que se vai utilizar. Dada a escolha do sistema operativo ter sido exclusivamente *Android* fica-se restringido às

opções de *Java* ou *Kotlin*. Uma vez que o *SDK* para *Android Java* surgiu primeiro e, consequentemente, tem uma maior comunidade por trás, além de haver uma maior familiaridade com a linguagem *Java*, levaram à escolha desta linguagem.

Finalmente é então necessário escolher a ferramenta com que se vai desenvolver a componente de SIG. A condição principal para a escolha da ferramenta a utilizar é que esta possua um *SDK*. Um *SDK* consiste num conjunto de ferramentas, bibliotecas, exemplos de código e documentação que permitem desenvolver *software* para plataformas específicas. Desta forma é possível integrar a componente de SIG com a ajuda destas ferramentas em conjunto com outras componentes na *app*, como por exemplo, a respeitante às funcionalidades de *BLE*.

Foram analisadas várias opções que serão descritas de seguida em conjunto com a opção escolhida.

- *CARTO*

CARTO é uma plataforma *open-source* e *cloud-based* direcionada para a visualização e análise espacial de dados de localização através de uma interface intuitiva baseada em *widgets drag-and-drop* que dispensa os utilizadores de conhecimento na utilização de SIG.

Esta facilidade permite desenvolver aplicações de localização altamente personalizáveis e sem a necessidade de conhecimentos de programação num curto espaço de tempo, o que confere a analistas de negócios, cientistas de dados ou até utilizadores individuais a capacidade de extrair informação e tomar decisões mais sustentadas com base nos dados de localização.

Esta plataforma dispõe de uma ferramenta denominada *Builder* que agrega dados de diversas fontes como folhas de cálculo, dados provenientes de redes de sensores, de outros softwares de negócio e até de *datasets* disponíveis na plataforma permitindo fazer um conjunto de operações como filtragem, enriquecimento de dados e aplicação de análise espacial e temporal.

A plataforma também conta com um *SDK* que permite desenvolver aplicações para diversas plataformas de entre as quais *Android*. Para além disso conta com um conjunto de *APIs* que permitem fazer uma série de ações como por exemplo autenticação na plataforma (*Auth API*), importar ficheiros para um conta CARTO (*Import API*), interagir com tabelas e dados alojados numa conta CARTO (*SQL API*), usar uma série de serviços de localização como geocoding (*Data Services API*) e gerar mapas com base em dados alojados numa conta CARTO (*Maps API*).

As principais caraterísticas desta plataforma são:

- Abstrai o utilizador da complexidade das ferramentas *SIG*
- Interface *user-friendly*
- Importar *datasets* provenientes de múltiplas fontes inclusive os disponibilizados pela plataforma
- Um conjunto de *API's* que permite interagir com a conta CARTO
- *SDK* para desenvolvimento de aplicações para diversas plataformas

Ainda que esta plataforma seja originalmente *open-source*, estando disponível o seu código-fonte num repositório no *GitHub*, a sua utilização está condicionada ao pagamento de uma subscrição de um dos dois planos disponíveis: *Professional* ou *Enterprise*.

De modo a se evitar pagar esta subscrição seria necessário assegurar a implementação de todas as funcionalidades já disponibilizadas na versão paga, implicando ainda assim que se perdesse o acesso ao suporte conferido pelo pagamento de uma das subscrições.

Assim sendo, a versão implementada e paga do CARTO acaba por se tornar uma forma conveniente e fácil de se ter acesso à tecnologia já disponível de forma open-source.

De igual forma o *SDK* para desenvolvimento de aplicações *mobile* e *tablet* continua disponível na versão implementada no entanto com a condicionante de se ter que desenvolver não só a aplicação como todas as restantes

funcionalidades para alojar o mapa e os *datasets* gerados localmente assegurando a correta integração entre os vários componentes do sistema.

Por último conclui-se que esta opção, de forma a se tornar pouco dispendiosa (o que requer não optar pela versão paga), implicaria um grande investimento de tempo no desenvolvimento e implementação de todas as componentes do sistema.

- MapBox

O MapBox é uma plataforma *open-source* de localização que permite criar mapas esteticamente elaborados e dados de localização intuitivos.

A plataforma disponibiliza uma ferramenta denominada *Studio* que permite criar mapas altamente personalizáveis a partir de uma série de mapas-base disponíveis dando a liberdade de personalizar livremente um conjunto alargado de caraterísticas do mapa, *tilesets* que são um conjunto de dados do tipo vetorial ou raster e datasets que são um conjunto de *features* do tipo *GeoJSON* .

Para além disso também disponibiliza um conjunto de *API's* divididas em quatro serviços e *SDK's* que permitem incorporar as funcionalidades do MapBox em aplicações web e mobile.

Conta também com a funcionalidade de navegação que permite ter acesso a tráfego em tempo-real, orientações dependentes do tráfego, acidentes, etc. e conta com uma API robusta de geocoding direta e indireta que converte coordenadas em morada e vice-versa.

As principais caraterísticas desta plataforma são:

- Sem necessidade de ter conhecimentos em *SIG*
- Interface intuitiva
- Elevado nível de personalização do mapa
- Permite importar *datasets* ou *tilesets* a partir de vários formatos de ficheiros.
- Uma série de *API's* que permite interagir com a conta *MapBox*

- *SDK's* para desenvolver aplicações web ou mobile para várias plataformas

À semelhança do *CARTO*, o *MapBox* também é uma plataforma *open-source* com o seu código-fonte disponível num repositório no *GitHub* dispondo também de uma componente paga.

No caso desta plataforma as opções de pagamento dividem-se entre suporte havendo três subscrições que conferem níveis de apoio diferentes, a nível dos *SDK's* para desenvolvimento *mobile* em que há a cobrança a partir de 25,000 utilizadores, a nível de carregamento dos mapas *Web* havendo a cobrança a partir de 50,000 carregamentos e finalmente a nível das *API's* existindo também um limite para cada uma a partir do qual se efetua a cobrança.

Apesar das semelhanças entre estas duas plataformas, o *MapBox* apresenta-se como uma solução financeiramente mais acessível uma vez que o plano gratuito oferece uma margem significativa de utilização das funcionalidades desta plataforma.

No entanto, durante a fase de escolha das opções tecnológicas para a implementação da componente *SIG*, testou-se o *SDK* desta plataforma e verificou-se que com a atualização para uma nova versão do *SDK* ficaram descontinuadas muitas das funções previamente disponíveis, não se tendo encontrado uma boa base de documentação relativa à transição de versões.

Além disso constatou-se na leitura da documentação que era possível descarregar mapas para utilização *offline* no entanto não era possível modificá-los e sincronizar as alterações posteriormente. É sim possível descarregar *datasets* podendo-se fazer alterações e sincronizar com recurso à *API* apropriada (a *Datasets* API), no entanto as alterações seriam apenas no *dataset* e não no mapa que o continha.

Por estas razões acabou-se por se descartar esta solução para a concepção do *SIG*.

- *Google Maps*

O *Google Maps* é uma aplicação de serviços de cartografia concebida pela *Google* que é disponibilizada de forma gratuita (para licenças não-comerciais) e que é usada em vários serviços de mapas como o próprio *website* do *Google Maps* e os *websites* que incorporam as suas funcionalidades através da *Google Maps API.*

Esta aplicação fornece serviços de pesquisa e visualização de mapas e imagens de satélite assim como uma componente de navegação que pode ser utilizada em trajetos feitos por carro, transportes públicos e a pé e uma componente para localizar pontos de interesse, empresas, etc.

De forma a se poder integrar as funcionalidades do *Google Maps* em aplicações é disponibilizado um *SDK* que contém bibliotecas que interagem com a *Google Maps API*.

Para a utilização desta *API* e de outras disponíveis é necessário utilizar uma chave de *API* que é gerada na *Google Cloud Platform* e que permite não só identificar a aplicação que a utiliza mas também controlar o seu acesso e cotas atribuídas.

Além desta *API* estão também disponíveis outras que permitem fazer operações como obter direções entre localizações (*Directions API*), distância e tempo de viagem para múltiplos destinos (*Distance Matrix API*), converter moradas em coordenadas geográficas e vice-versa (*Geocoding API*), entre outras *APIs* disponíveis. Cada uma destas *APIs* apresenta uma tabela de preços que dispõe de uma utilização gratuita até um determinado número de chamadas em que a partir das quais se inicia a cobrança. A utilização das *APIs* pode ser controlada através da *Google Cloud Platform*.

As principais caraterísticas desta plataforma são:

- Sem necessidade de ter conhecimentos em *SIG*
- Interface intuitiva
- Existência de um plano gratuito com um determinado *plafond*

- Uma série de *API's* que permite incorporar funcionalidades interessantes de navegação.
- *SDK's* para desenvolver aplicações web ou mobile para várias plataformas.
- Boa documentação para a utilização das *API's* e *SDK's*

Ainda que esta solução apresente funcionalidades interessantes com um *plafond* gratuito que as permite explorar numa utilização não muito intensiva, apresenta no entanto bastantes limitações que a fizeram ser desconsiderada.

Em primeiro lugar não cumpre adequadamente um dos requisitos essenciais do projeto desta dissertação: permitir a aplicação trabalhar *offline*. De forma a contornar esta limitação é possível implementar-se uma *TileProvider* personalizada. Uma *TileProvider* fornece as imagens a serem usadas numa camada que se comporta como mapa-base. Para que se pudesse utilizar a *app* inteiramente *offline* seria necessário descarregar todas as imagens desta camada no momento da instalação o que aumentaria consideravelmente o espaço que a *app* ocuparia no armazenamento.

Além disso, esta solução também não permite exibir ficheiros do tipo *ESRI shapefile* e apresenta grandes limitações a nível do tamanho de ficheiros KML e GeoJSON.

Por estas razões a opção do *Google Maps* foi descartada para desenvolvimento da componente *SIG.*

- *QGIS*

O *QGIS* é um *software* de sistema de informação geográfica *open-source* e multi-plataforma que permite aos utilizadores fazer uma série de operações como criar, ver, editar, e analisar dados geo-espaciais assim como sobrepor camadas do tipo vetorial ou *raster* numa série de formatos e projeções.

Para além disso, contém também um conjunto de funcionalidades e ferramentas típicas de *SIGs* como fazer análise espacial de dados (que pode também

ser feita em bases de dados geo-espaciais), análise vetorial, geoprocessamento, gestão de base de dados, entre outras.

Este *software* suporta vários tipos de formatos como *ESRI shapefiles*, bases de dados geo-espaciais como a *PostgreSQL* com a extensão *PostGIS* e serviços *Web* como *Web Map Service (WMS)* e *Web Feature Service (WFS)* que permitem aceder a fontes externas de dados.

Para complementar as ferramentas já existentes no *software* existem mais de 1000 *plugins* disponíveis que permitem potenciar ainda mais o *QGIS* com novas funcionalidades consoante as necessidades da aplicação. Além disso a criação de novos *plugins* é facilitada usando linguagens como *Python* ou *C++.*

Finalmente esta solução conta também com um *SDK* para a plataforma Android disponível num repositório *GitHub* permitindo assim desenvolver soluções que recorram às potencialidades desta tecnologia.

As principais caraterísticas desta plataforma são:

- Solução *open-source* que por si só não tem custos
- Documentação muito boa e uma comunidade ativa
- Existência de mais de 1000 *plugins* que potenciam ainda mais o *software.*
- *SDK's* para desenvolver aplicações *mobile* para *Android*

Ainda que esta solução permita desenvolver todas as funcionalidades necessárias possivelmente sem qualquer custo, existem algumas limitações que fizeram com que não fosse escolhida.

Em primeiro lugar, apesar de existir um *SDK* para desenvolver aplicações para a plataforma *Android*, o que foi constatado é que não há muita documentação nem comunidade ativa na utilização deste *SDK* comparativamente à solução *QGIS Desktop* e à versão *mobile* denominada *QField*. Além disso ao optar-se por esta solução implicaria desenvolver diversas componentes de raíz como por exemplo, a base de dados local do *smartphone/ tablet*, a base de dados alojada num servidor na *Web* contendo toda a sua informação dos mapas como *features*

existentes, e um servidor para fornecer os mapas e as diversas *layers* de informação. Isto implicaria um grande investimento de tempo e dada a janela temporal existente para a realização da presente dissertação poderiam não ser cumpridos os requisitos propostos atempadamente. Assim sendo não se optou por esta solução.

- **ArcGIS**

O ArcGIS é uma plataforma que permite criar, visualizar, analisar, gerir e partilhar dados espaciais de forma a compreender o seu contexto geográfico e identificar padrões entre eles. Para tal a plataforma conta com diversos componentes que comunicam entre si através da ArcGIS REST API e formatos de ficheiros compatíveis.

Na base da plataforma encontra-se o ArcGIS Online e ArcGIS Enterprise que são complementados com outras componentes como: o ArcGIS Desktop, constituído pelo ArcGIS Pro e ArcMap, que permite publicar e gerir dados e mapas em conjunto com o ArcGIS Online e ArcGIS Enterprise podendo as suas capacidades serem extendidas com uma série de ferramentas; um conjunto de Apps disponibilizadas que trabalham também com o ArcGIS Online e ArcGIS Enterprise para consumir e disponibilizar dados; e um conjunto de API's e SDKs que podem ser usados para criar aplicações personalizadas nativas ou *web*-based que podem extender as aplicações disponbibilizadas com outras funcionalidades.

Na Figura 17 encontra-se representada a *dashboard* do ArcGIS Online onde é possível criar mapas e *layers* de informação.

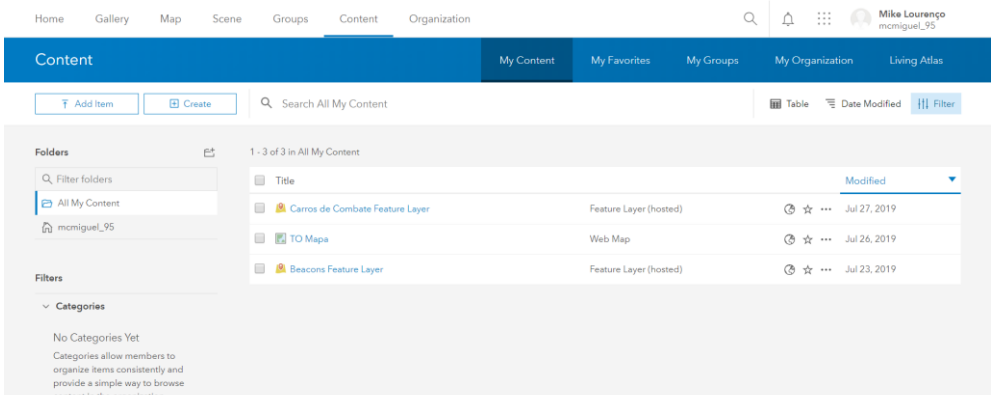

Figura 17 - *Dashboard* do *ArcGIS Online* que permite a criação e gestão de mapas e *layers* de informação

Esta plataforma permite recolher, processar e analisar dados provenientes de múltiplas fontes e em diversos formatos e projeções que, combinados com ferramentas de análise espacial disponibilizadas, conduzem à tomada de decisões informadas que poderão ter um impacto decisivo no sucesso de negócios.

Dispõe também de flexibilidade no armazenamento, edição e gestão de dados adaptados às necessidades do usuário permitindo também a fácil conexão com bases de dados relacionais (RDBMS) como, por exemplo, SQL Server e PostreSQL ou até na cloud da ESRI através do ArcGIS Online.

A disponibilização de mapas e informação geográfica poderá ser feita selectivamente. Uma das formas de disponibilização é através de uma organização que corresponde a um conjunto de membros de um grupo de trabalho comum.

A utilização desta plataforma tem custos associados que estão divididos sobre diversos planos consoante o produto escolhido. No âmbito do projeto desta dissertação, o produto a ser escolhido seria o ArcGIS for Developers que permite ter acesso aos SDKs e ao ArcGIS Online de forma gratuita ainda que com algumas restrições. Para este produto existem dois planos designados Essentials e Builder. O primeiro é gratuito e tem um plafond de utilização que abrange as necessidades da *app* a ser desenvolvida. O segundo plano tem um custo mensal associado e a diferença mais significativa face ao primeiro plano e, tendo em conta as necessidades da *app* a ser desenvolvida, é a possibilidade de desenvolver *apps* licenciadas que possam ser comercializadas, algo que não é possível com o plano Essentials que restringe a comercialização das *apps* desenvolvidas.

Para além destes planos, este produto carece também do pagamento do licenciamento da *app* para o caso desta vir a ser comercializada. Para o licenciamento existem quatro planos denominados Lite, Basic, Standard e Advanced sendo que só o plano Lite é gratuito e vem incluído com o plano Essentials.

Por último existe também um plano do tipo *pay as you go* em que se compra créditos antecipadamente para a utilização de *web services* do ArcGIS. Mensalmente é atribuído um *plafond* gratuito que permite explorar estes *web services* com restrições de utilização que permitem aceder a serviços de busca de localizações, *geocoding*, *routing* para obter direções para um destino, aceder a dados

demográficos, realizar análise espacial e utilizar o armazenamento na *cloud*. A utilização destes *web services* carece de um custo pré-definido que é descontado do crédito disponível. Caso a utilização ultrapasse o plafond será então necessário adquirir mais créditos para se poder continuar a dispor destas funcionalidades.

As principais caraterísticas desta plataforma são:

- Solução completa face às necessidades da componente SIG do projeto da dissertação, sem necessidade de desenvolver componentes adicionais.
- Documentação muito boa e muitos exemplos de *apps* disponíveis.
- *SDK's* para desenvolver aplicações *mobile* para *Android* e outras plataformas.
- ArcGIS for Developers com um plano gratuito que permite aceder a um vasto número de funcionalidades.

- **Plataforma eleita**

Após ser feita uma análise a todas as soluções examinadas concluiu-se que a plataforma ArcGIS seria a mais adequada para o desenvolvimento da componente SIG do projeto desta dissertação.

Ainda que seja comparativamente a solução mais cara de entre as analisadas, o que aparentemente poderá divergir do objetivo principal desta dissertação de criar um equipamento de baixo custo, a relação custo-benefício obtida com esta solução é significativamente superior.

Em primeiro lugar, optando pela plataforma ArcGIS em particular pelo plano Essentials do ArcGIS for Developers, tem-se acesso a um plafond gratuito de 50 créditos que permite uma utilização gratuita de muitas funcionalidades que colmatam grande parte das necessidades do sistema, não sendo por isso obrigatório partir de um plano que exija um pagamento mensal como é o caso do plano Builder. O pagamento poderá advir essencialmente da necessidade de licenciamento da *app* para fins comerciais e da utilização de *web services* que consumam

estes créditos como, por exemplo, o que permite obter uma rota até uma determinada localização (*routing*).

Além disso há que salientar que atualmente o Estado português e outras instituições nacionais recorrem a tecnologia da ESRI o que representa duas grandes vantagens a nível de adaptação e familiariadade com a tecnologia e a nível negocial podendo-se estabelecer acordos que permitam a sua utilização com custos mais competitivos.

Assim sendo, tendo por base estes benefícios e não haver necessidade de desenvolver a componente SIG de raíz, a plataforma ArcGIS e o seu produto ArcGIS for Developers foi a eleita para o desenvolvimento desta componente da *app.*

Com base nos requisitos propostos para a *app* a ser desenvolvida no âmbito desta dissertação constatou-se a necessidade de optar por desenvolvimento nativo sendo Android a plataforma escolhida e Java a linguagem a ser utilizada. Desta forma recorreu-se então ao ArcGIS Runtime SDK for Android disponível no ArcGIS for Developers que disponibiliza todas as ferramentas necessárias, em conjunto com uma vasta documentação e exemplos de aplicações, para o desenvolvimento da componente SIG da *app*.

3.4.2. Arquitetura da *app tablet*

3.4.2.1. Arquitetura geral da *app*

Escolhida a tecnologia de implementação da componente *SIG* e a linguagem com que se vai desenvolver a *app* é necessário agora definir a sua arquitetura detalhando os várias componentes e as suas interações.

O diagrama da Figura 18 representa o funcionamento de alto nível do sistema.

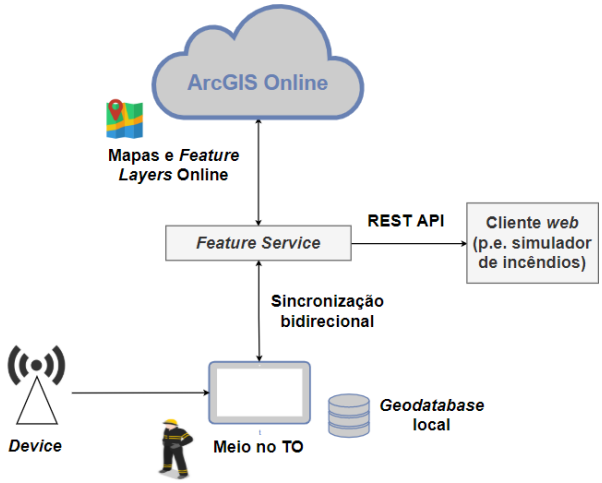

Figura 18 - Diagrama de alto nível do sistema

No centro do sistema situam-se os mapas e as *Feature Layers* desenvolvidas através do ArcGIS Online. A sua implementação será detalhada mais à frente na secção 3.5.

A partir do mapa criado com as *Feature Layers* de interesse foi exposto um *Feature Service* que fornece as *features* presentes nas *layers* criadas assim como tabelas não-espaciais (que contêm apenas dados de atributos) e a representação no mapa das *features* (a sua geometria). Através do *ArcGIS Online* é possível configurar as definições de partilha do *Feature Service* e assim definir quem pode aceder e editar as *features* presentes nas *Feature Layers* podendo assim criar-se grupos de trabalho que tenham acesso restrito aos mapas e *layers* de informação.

Ao expor as *Feature Layers* no *Feature Service* estas ficam disponíveis para serem consumidas por aplicações móveis, clientes *web* e aplicações *desktop* podendo assim serem utilizadas na aplicação a ser desenvolvida nesta dissertação.

Por outro lado também podem ser consumidas por clientes *web* que encaminhem os dados para, por exemplo, um simulador de propagação de fogos.

O consumidor principal dos dados da *Feature Service* será a aplicação *tablet* disponibilizada aos meios no terreno. De forma a poder operar *offline* a aplicação terá uma *geodatabase* (base de dados geográfica) local que será sincronizada com o *Feature Service* e permitirá assim ao utilizador não depender de uma conexão de rede. Todas as alterações sobre os dados serão feitas localmente e

armazenadas na *geodatabase* sendo sincronizadas assim que haja a conexão de rede e sobre a instrução do utilizador.

É importante referir que a sincronização é bidirecional sendo as alterações mais recentes as que são registadas, pelo que se houver um segundo utilizador a trabalhar no mapa no *ArcGIS Online*, as suas alterações poderão ser substituídas por um utilizador da aplicação *tablet* que esteja a trabalhar no mesmo.

Finalmente temos o *device* cuja implementação foi descrita na secção 3.3 que comunica com a aplicação *tablet* através das interfaces de comunicação *Bluetooth BLE* ou *Wifi* (sendo a interface *BLE* a priveligiada) e que é representado no mapa como uma *feature* do tipo *Point* (um ponto) cujos atributos são os dados sensoriais transmitidos. Esses dados serão depois sincronizados com a *Feature Service* e poderão assim serem consumidos por um cliente *web* que os encaminhe, tal como foi referido anteriormente, para um simulador de incêndios ou apenas visualizados no *ArcGIS Online*.

A aplicação *tablet* terá então que ter um conjunto de funcionalidades que incluam as típicas de um *SIG* e que possibilitem a comunicação com um dispositivo externo que forneça dados de interesse.

No diagrama da Figura 19 encontram-se representadas as várias componentes da aplicação *tablet*.

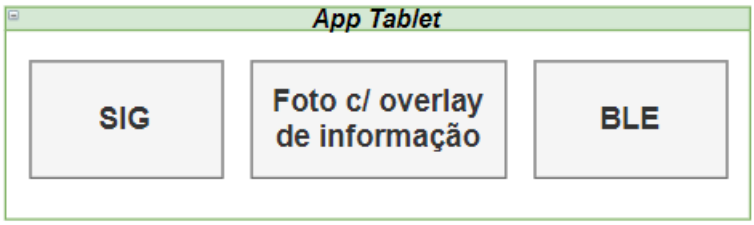

Figura 19 - Diagrama representativo das componentes da app

A *app* será então constituída essencialmente por três componentes que serão descritas de seguida:

- ***SIG*** – respeitante ao Sistema de Informação Geográfica onde serão desenvolvidos os requisitos respeitantes à criação, modificação e remoção de *features*, anexação e remoção de anexos provenientes da câmera ou da galeria, criação duma base de dados local (*geodatabase*) e sincronização com o *Feature Service online* (que detém as *layers* de informação com a sua simbologia associada e tabelas com os seus atributos).

- **Foto c/ *overlay* de informação** – está integrada na opção de adicionar anexos na componente *SIG*; permite sobrepor dados de interesse sobre uma foto como, por exemplo, uma *timestamp* de quando foi captada e a sua localização geográfica.

- ***BLE*** – permite fazer a pesquisa de dispositivos *Bluetooth BLE* e estabelecer conexão; responsável também pela gestão da conexão em segundo plano com o *device* e leitura das notificações por ele geradas com os parâmetros sensoriais mais recentes.

As classes Java desenvolvidas, responsáveis por implementar estas componentes, são as seguintes:

- ***MainActivity*** - classe principal; nesta classe é tratada a interação com o mapa e com a *ActionBar* e são recebidos os dados provenientes do serviço que gere a ligação *Bluetooth BLE*; aqui estão também os métodos da componente *SIG* e as *AsyncTasks* que fazem a compressão das imagens em segundo plano.
- ***DeviceScan*** – nesta classe é iniciada a procura por dispositivos *Bluetooth BLE*; os resultados são apresentados numa *ListView;* o utilizador seleciona o dispositivo a que se quer conectar e é retornado para a *MainActivity* o nome e endereço do dispositivo selecionado.
- ***BluetoothLeService*** – classe que implementa um serviço responsável por gerir a ligação *Bluetooth BLE* com o *device* em segundo plano; aqui estabelece-se a conexão com o *device* e são recebidos os dados

das *Characteristics* subscritas que são posteriormente encaminhados para a *MainActivity*.

- ***BitmapCompressionParams*** – classe que agrega os parâmetros de entrada das duas *AsyncTasks* num só objeto disponibilizando duas declarações distintas para cada uma das AsyncTasks.

Com a arquitetura geral definida segue-se então para a descrição detalhada do processo de implementação de cada uma das componentes nas próximas secções.

3.4.2.2. Implementação da componente *SIG*

A implementação da componente *SIG* dividiu-se em várias fases que serão descritas de seguida.

Após ser estabelecida a arquitetura da componente *SIG* é necessário ler a documentação para a correta utilização do *ArcGIS Runtime SDK* para *Android*.

O *website* do *ArcGIS for Developers* dispõe de documentação muito completa para a utilização deste *SDK* contando com tutoriais que permitem uma familiarização rápida com a tecnologia e desenvolvimento de aplicações-exemplo.

Em primeiro lugar foi necessário instalar o *SDK,* o que foi feito com recurso a um tutorial existente que detalha como o fazer utilizando a ferramenta *Gradle*, uma ferramenta *open-source* que, no contexto do desenvolvimento de aplicações *Android*, permite automatizar o processo de compilação, criação e *packaging* de aplicações ou bibliotecas *Android*.

Estando concluída a instalação do *SDK* passou-se ao próximo passo que visa a exibição de um mapa e que será o ponto de partida para o desenvolvimento das funcionalidades da componente *SIG*.

- **Exibição de um mapa da *ESRI***

Para se poder mostrar um mapa da *ESRI* na aplicação é necessário incluir o elemento *XML* do *MapView* no ficheiro de *layout* de uma *Activity*. O excerto de código que permite incluir a *MapView* está representado na Figura 20.

```
<!-- MapView -->
    <com.esri.arcgisruntime.mapping.view.MapView
        android:id="@+id/mapView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentBottom="true"
        android:layout_alignParentStart="true"/>
```

Figura 20 - Elemento XML que inclui a MapView no layout da aplicação

O ficheiro de *layout* é um ficheiro *XML* localizado na pasta *res/layout* que define a estrutura de uma interface de utilizador numa aplicação e que pode estar associado a uma *Activity*.

Os elementos base de uma interface de utilizador são os objetos *View* que são criados recorrendo à classe *View* e ocupam uma área retangular no ecrã sendo responsáveis por fazer aparecer conteúdo interativo na interface de utilizador. *View* é também a classe base para os *widgets* que são usados para criar componentes interativos na interface de utilizador como botões, caixas de texto, entre outros.

Um nível acima existem os *ViewGroups* que é um tipo especial de *View* também designado *container* que pode conter outras *Views* ou *ViewGroups* e definir as suas propriedades de *layout*.

Os *layouts* são sub-classes da classe *ViewGroup* e podem ser criados durante a execução do programa recorrendo a objetos *View/ViewGroup* ou através de um ficheiro *XML* tal como foi descrito acima.

A maioria dos ficheiros *XML* de *layout* presentes na pasta *res/layout* estão associados a *Activities* onde definem a interface de utilizador da mesma. Uma *Activity* constitui um componente de uma aplicação que exibe uma interface de utilizador e tem funcionalidades associadas. Esta tem um ciclo de vida composto

por vários estados a que estão associados métodos de retorno de chamada (*call-backs*) que são chamados aquando das mudanças de estado. Alguns exemplos de mudanças de estado podem ser a criação da *Activity*, a sua interrupção, o seu retorno e a sua destruição.

Na Figura 21 está representado o ciclo de vida de uma Activity.

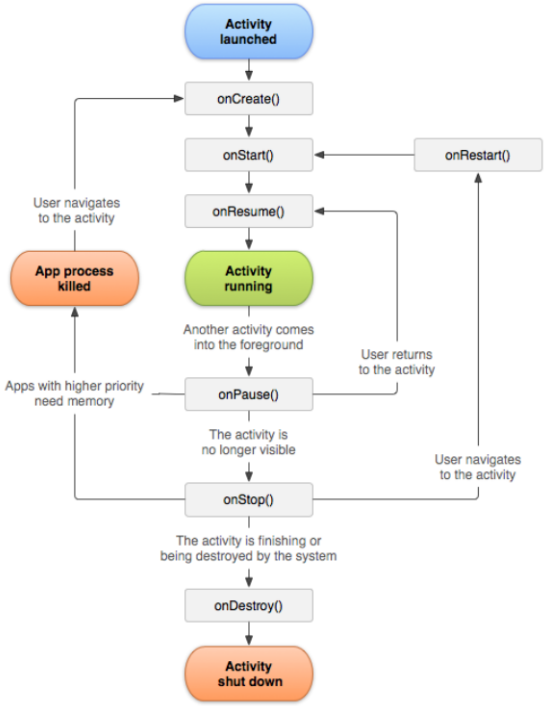

Figura 21 - Ciclo de vida de uma Activity

Um dos métodos de *callback* mais importante é o *onCreate()* que é chamado no momento de criação da *Activity*. Deve-se implementar este método e é nele que se deve fazer a inicialização dos componentes essenciais da *Activity* como criação de *Views*, inicialização de variáveis, etc. É neste método também que se deve chamar o método *setContentView()* que permite definir o *layout* para a interface de utilizador da *Activity* passando como parâmetro a *View* ou o identificador do ficheiro *XML* onde o *layout* está definido.

Um dos componentes que deve ser inicializado no *onCreate()* é o *MapView* e mapa a ser mostrado nele. O *MapView* é responsável por processar e mostrar

os dados de um *ArcGISMap* num *layout* e permite aos utilizadores interagir com o mapa.

A Figura 22 mostra como configurar o *MapView* para que este carregue o mapa, o seu mapa-base, *layers* operacionais e mostre os seus conteúdos no ecrã.

```
MapView mMapView = findViewById(R.id.mapView);
ArcGISMap map = new ArcGISMap(Basemap.Type.Topographic, 32.024785, -9.542147, 15);
mMapView.setMap(map);
```

Figura 22 - Excerto de código para configurar e mostrar o mapa no ecrã

Por definição o *MapView* suporta gestos padrão de interação com o mapa como fazer *zoom*, rodar o mapa, toque duplo, entre outros. No entanto é também possível implementar os métodos associados a estes gestos para que se possa configurar a aplicação para responder da forma desejada a estas interações.

Na aplicação desenvolvida foram implementados os métodos: *onSingleTapConfirmed()* e *onLongPress()* para desenvolver certas funcionalidades. A sua implementação será explicada mais à frente.

- Criação e carregamento da *geodatabase* para funcionamento *offline*

De forma a poder utilizar-se a aplicação *offline* e sincronizar as alterações feitas assim que a conectividade seja restabelecida, uma das estratégias que se pode adoptar é a de utilização de uma *geodatabase* que é armazenada localmente sobre a forma de um ficheiro.

Um ficheiro *geodatabase* é na prática uma cópia dos dados presentes num *ArcGIS service* que permite armazenar, consultar e gerir dados espaciais e não-espaciais. Este contém uma ou mais *GeodatabaseFeatureTables* que possuem dados de atributos e geometria das *features.* Uma *GeodatabaseFeatureTable* contendo *features* do tipo *Point, Polyline* ou *Polygon* pode ser convertida para *Feature Layer* e ser carregada no mapa como *layer* operacional para visualização e interação.

Na implementação da aplicação é verificado no início a existência de um ficheiro *geodatabase* para carregamento dos dados nele presente. Caso exista, procede-se ao carregamento em memória dos dados, caso contrário é criada a *geodatabase*. Esta verificação é efetuada na função *openOrCreateGeodatabase()*.

O diagrama da Figura 23 representa esquematicamente este processo.

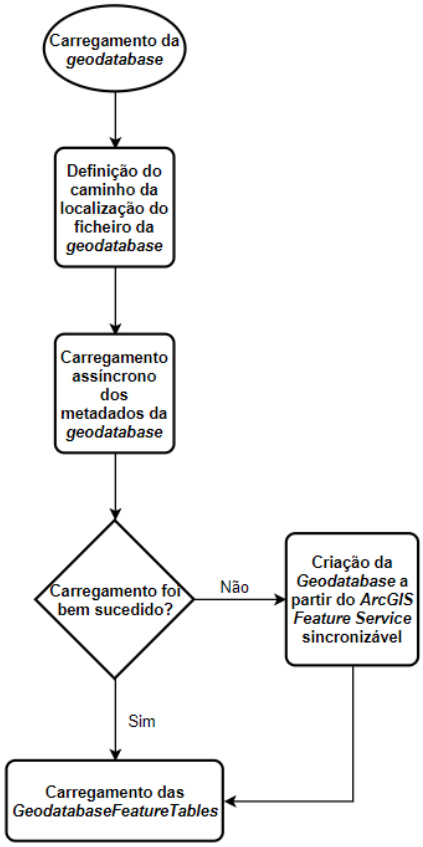

Figura 23 - Fluxograma do funcionamento da função openOrCreateGeoDatabase()

Inicialmente é estabelecido um caminho para a localização do ficheiro no armazenamento local do dispositivo com o qual é criado um objeto do tipo *Geodatabase*.

Posteriormente, a informação àcerca da *geodatabase* (metadados) é carregada assincronamente o que é possível dado que a classe *Geodatabase* implementa o padrão de design *Loadable* que permite informar sobre o sucesso de

operações assíncronas de carregamento de metadados e fornecer um mecanismo de repetição em caso de falha.

É então adicionado um *listener* através do método *addDoneLoadingListener()* que é executado quando o carregamento é concluído. Assim que isso aconteça, verifica-se o estado do carregamento e em caso de que este tenha sido bem sucedido, significa que a *geodatabase* existe e a sua informação foi corretamente carregada pelo que se pode proceder ao carregamento da *geodatabase* que é efetuado na função *loadingGeodatabase().*

Na função *loadingGeodatabase()* começa-se por fazer o carregamento dos metadados das suas *GeodatabaseFeatureTables*. Após esse carregamento ter sido bem sucedido, a partir de cada *GeodatabaseFeatureTable* são criadas *FeatureLayers* que se adicionam ao mapa como *layers* operacionais. Os atributos de cada *GeodatabaseFeatureTable* são também guardados para serem usados mais tarde.

Após ser feito o carregamento da *geodatabase*, o utilizador é questionado se deseja fazer a sincronização com o *Feature Service*. Tal acontece pois desde a última sincronização poderão ter havido alterações feitas através de outro utilizador da aplicação ou por um utilizador do ArcGIS Online que esteja a trabalhar nos mesmos mapas.

Se o utilizador quiser sincronizar será chamada a função *syncGeodatabase()*. Nesta função começa-se por criar um objeto do tipo *SyncGeodatabaseParameters* onde são configurados os parâmetros de sincronização. O parâmetro configurado foi a direção de sincronização, através da função *setSyncDirection()* em que se configurou para que esta fosse bidirecional de modo a sincronizar as alterações mais recentes entre as do dispositivo e as do *Feature Service*. Neste caso, se as alterações mais recentes fossem as do dispositivo seriam estas as que seriam aplicadas no *Feature Service* ao passo que, se fossem as do *Feature Service* as mais recentes, seriam então essas as aplicadas no dispositivo.

De seguida foram criados e adicionados objetos do tipo *SyncLayerOption* que continham os IDs das *GeodatabaseFeatureTables* presentes na *Geodatabase*

aos parâmetros de sincronização de modo a incluir as *GeodatabaseFeatureTables* na sincronização.

Finalmente foi criado um objeto *SyncGeodatabaseJob* a partir dos parâmetros de sincronização configurados e da *geodatabase* que se pretende sincronizar ao qual se chamou a função *start()* para iniciar o processo de sincronização. No final deste processo foi guardada a data e hora da sincronização.

Por outro lado, caso o carregamento dos metadados da *geodatabase* não tenha sido bem sucedido implica que o seu ficheiro não existe pelo que se procede à criação do mesmo na função *generateGeodatabase().* Nesta função começa-se por criar um objeto do tipo *GeodatabaseSyncTask* com o *URL* do *Feature Service* cujos dados se quer guardar localmente.

De seguida é criado um objeto do tipo *GenerateGeodatabaseParameters* onde se especifica a extensão do mapa que se quer trabalhar e se queremos que se inclua anexos. Nesta aplicação será de interesse incluir-se anexos onde se poderão ter, por exemplo, fotos de frentes de incêndio.

Posteriormente é criado um objeto do tipo *GenerateGeodatabaseJob* de onde se chama o seu método *start()* para que se peça ao *Feature Service* para gerar um ficheiro *geodatabase*.

Finalmente procede-se então ao carregamento da *geodatabase* o que é feito recorrendo à função *loadingGeodatabase().*

Com a *geodatabase* criada é agora possível trabalhar desconectado da rede e fazer a sincronização das alterações assim que esta seja restabelecida. As funcionalidades implementadas têm em conta operações registadas na *geodatabase* que possam ser sincronizadas posteriormente para o *Feature Service*.

Na Figura 24 está representado o arranque da *app* em que o utilizador é questionado se quer criar ou abrir uma *geodatabase* local e se pretende, em caso de abrir, sincronizar com o *Feature Service*

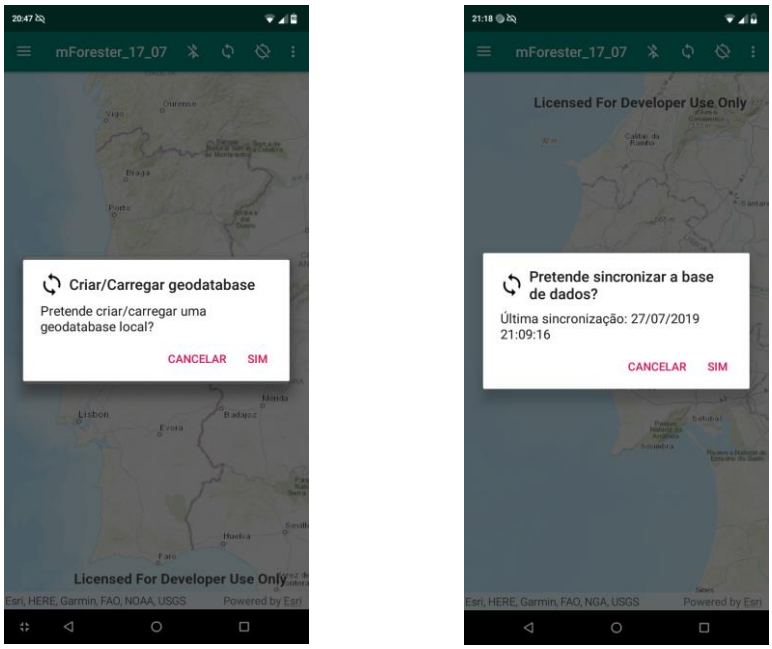

Figura 24 - Janela para criar/abrir uma geodatabase local e sincronizar a *geodatabase* com o *Feature Service*

- **Funcionalidade adicionar *feature***

A implementação da funcionalidade de adicionar uma *feature* ao mapa faz uso das capacidades de interação do *MapView.*

Como explicado anteriormente, existem funções de *callback* que são chamadas consoante o tipo de interação com o *MapView* e que podem ser implementados para a aplicação responder da forma que se deseje a estas interações.

Nesta funcionalidade foi usada a função de *callback onLongPress()* que é chamada quando o utilizador permance com o dedo no ecrã.

A função *onLongPress()* recebe como parâmetro uma variável do tipo *MotionEvent* que contém as coordenadas do ponto no ecrã que foi pressionado e que será utilizada para posicionar a *feature* criada no mapa.

Quando o utilizador pressiona o ecrã num ponto do mapa e mantém o dedo, a função *onLongPress()* é chamada e ele é questionado sobre a *layer* em que quer adicionar a *feature* que vai criar. Na aplicação realizada existem três

layers de três tipos diferentes: *Point*, *Polyline* e *Polygon.* Cada um deles terá uma forma específica de ser adicionado ao mapa.

O fluxograma da Figura 25 representa o processo para cada um dos tipos de *features.*

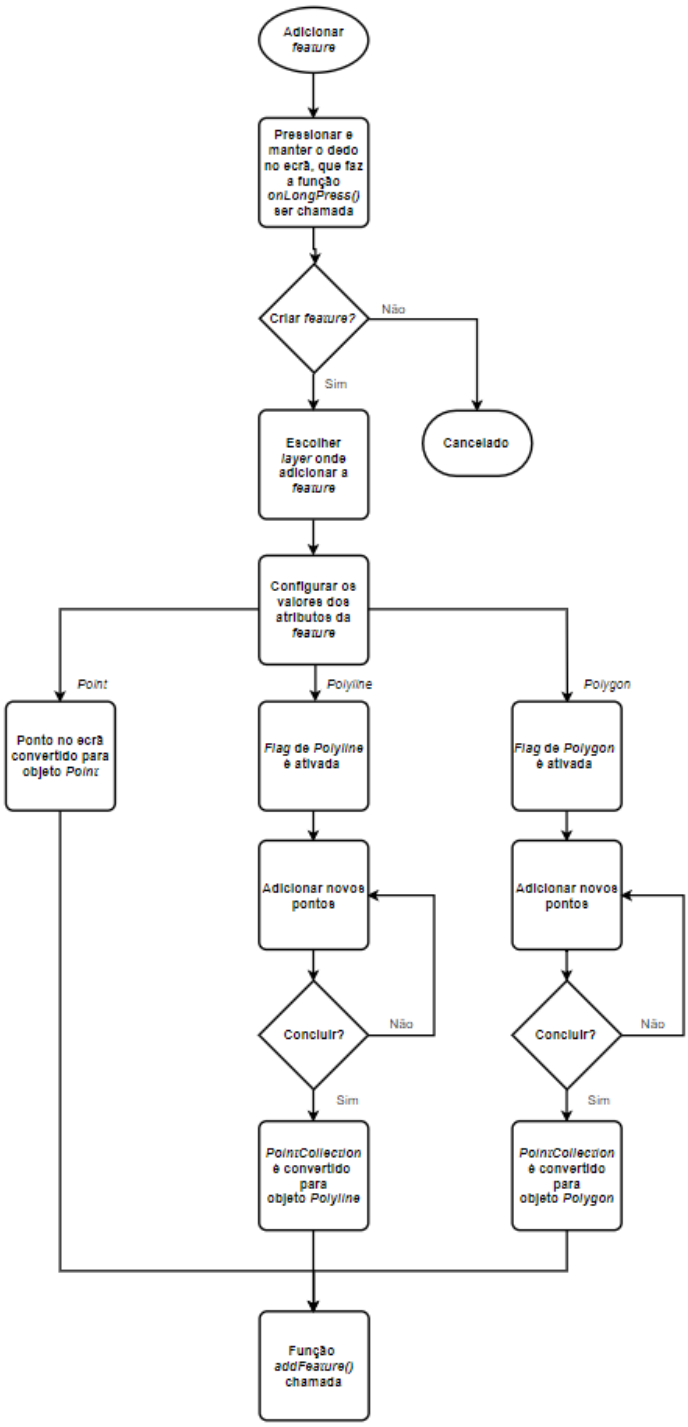

Figura 25 - Fluxograma representativo do processo de adição de features para os três tipos de geometrias: Point, Polyline e Polygon

Após a escolha da *layer* ser efetuada, será exibida uma janela com os atributos relativos à *layer* que o utilizador escolheu. Nesta janela o utilizador tem a opção de dar valores a cada um dos atributos sendo, no entanto, facultativo. A partir do momento em que os valores para os atributos sejam configurados, o processo seguinte dependerá do tipo de *feature* que o utilizador tenha escolhido.

No caso das *features* do tipo *Point*, após serem configurados os valores dos atributos, o ponto obtido através da variável *MotionEvent* é convertido numa variável do tipo *Point* que possa ser adicionada à *FeatureTable.* De seguida é chamada a função *addFeature()* que recebe como parâmetros a variável *Point* criada, o nome da *layer* em que deve ser adicionada e os atributos da *feature* previamente atribuídos.

Por outro lado, as *features* do tipo *Polyline* e *Polygon* têm um processo semelhante entre si de serem adicionadas. Após se terem configurado os valores dos atributos é ativada uma das *flags* atribuídas aos tipos *Polyline* e *Polygon,* que indicam que se está a adicionar um destes tipos de geometria. Simultaneamente é também preparado um objeto do tipo *PointCollection* que irá receber os pontos de cada uma das geometrias. Ambas as geometrias são um conjunto de pontos sendo as únicas diferenças o facto de que uma *Polygon* é uma geometria fechada cujo último ponto corresponde ao primeiro. Além disso é também ativada uma opção no menu que permite concluir a inclusão de pontos e criar a geometria.

A partir desse momento, cada ponto que o utilizador pressionar no ecrã será adicionado ao objeto *PointCollection*. Para tal, na função *onSingleTapConfirmed()* são verificadas as *flags* correspondentes a estes dois tipos de geometrias e caso alguma delas esteja ativa, o ponto pressionado é então adicionado ao *PointCollection* e a geometria no ecrã é atualizada com ele.

Durante o processo de inclusão de pontos é também possível remover o último ponto adicionado pressionando a opção "Retroceder" do menu.

Quando a geometria estiver concluída o utilizador pressiona a opção "Concluir" do menu que converterá o objeto *PointCollection* numa das geometrias pretendidas, *Polyline* ou *Polygon* que será posteriormente adicionada à *geodatabase* com recurso à função *addFeature()*.

A função *addFeature()* foi implementada de forma a poder adicionar os três tipos de geometrias à *geodatabase* e assim evitar três declarações distintas da mesma função. O primeiro argumento da função, que poderá ser um *Point, Polyline* ou *Polygon*, é recebido como variável do tipo *Object* e só dentro da função é que é criado um objeto do tipo *Feature* com os atributos passados consoante a geometria da *feature*. O segundo argumento da função é o nome da *FeatureTable* da *geodatabase* à qual se vai adicionar a *feature* e o terceiro argumento são os atributos pertencentes à *feature.*

Posteriormente, é adicionado o objeto *Feature* à *FeatureTable* da *geodatabase*. Esta operação é assíncrona e assim que esteja concluída o utilizador é notificado do seu sucesso ou insucesso.

Na Figura 26 está representado o menu que aparece quando o utilizador fica a pressionar uma zona do mapa e decide adicionar uma *feature*. Neste menu o utilizador decide a *layer* a que quer adicionar uma *feature*, sendo que a *layer "Beacons_Feature_Layer"* permite adicionar uma *feature* do tipo *Point,* a *layer "Linhas"* uma *feature* do tipo *"Polyline"* e a *layer "Áreas_Queimadas"* uma *feature* do tipo *Polygon.*

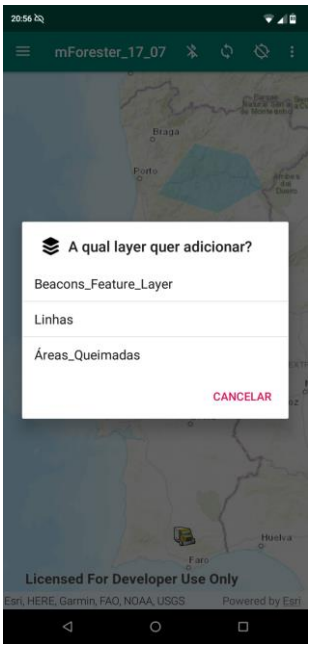

Figura 26 - Menu de seleção da layer a adicionar a feature

- **Funcionalidade ler atributos da *feature***

A implementação da funcionalidade de ler os atributos de uma *feature* recorre à função de *callback onSingleTapConfirmed().*

Quando o utilizador seleciona uma *feature* no mapa é exibido um *Callout* com uma série de opções disponíveis. Um *Callout* é um objeto obtido através do *MapView* que permite mostrar uma *View* com conteúdo como texto, imagens, etc.

Posteriormente é selecionada a opção que permite ler os atributos da *feature,* o que despoleta a chamada da *callback ImageViewOnclickListener()* onde é verificada a opção escolhida. Para o caso da opção escolhida ser a primeira, que corresponde à de leitura dos atributos da *feature*, é criado um objeto do tipo *Map<String, Object>* que se trata de um mapa *key-value pair* que contém os atributos da *feature* obtidos através da função *getAttributes()* e cujas *keys* são os nomes dos atributos e *values* os seus valores.

De seguida é criado um objeto *List<Field>* que corresponde a uma lista contendo os *Fields* (atributos) da *feature* selecionada.

Finalmente é criada uma caixa de diálogo do tipo *AlertDialog* que mostra uma mensagem contendo os atributos e os seus valores. Para tal são usados os nomes dos *Fields* para se extraírem os seus valores do mapa anteriormente obtido pela função *getAttributes().*

Na Figura 27 está representado o funcionamento desta funcionalidade.

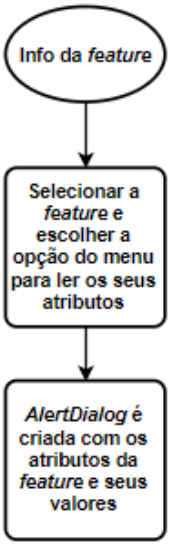

Figura 27 - Fluxograma representativo da opção de ler os atributos de uma feature

Na Figura 28 está representada a janela criada com os atributos da *feature* selecionada pelo utilizador.

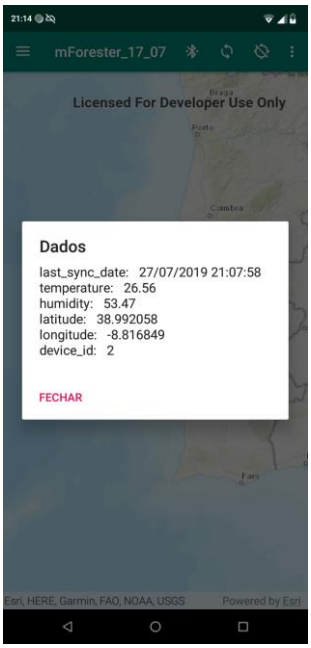

Figura 28 - Janela com os atributos da feature selecionada

- **Funcionalidade editar atributos de *feature***

A funcionalidade de editar os atributos recorre também à função *onSingleTapConfirmed()* sendo acessível através das opções exibidas no *Callout.*

Quando a *feature* cujos atributos se querem editar é selecionada, é exibido o *Callout* com as opções disponíveis.

De seguida é selecionada a opção que permite editar os atributos da *feature,* o que despoleta a chamada da *callback ImageViewOnclickListener()* onde é verificada a opção escolhida. No caso da opção escolhida ser a segunda, correspondente à da edição de atributos, são criados dois objetos do tipo *HashMap<String, EditText>* e *HashMap<String, TextView>* que irão conter, respetivamente, um número variável de *Views* para inserir o valor dos atributos (*EditText*) e *Views* para exibir o nome dos atributos (*TextView*). Desta forma a aplicação fica preparada para lidar com *features* com qualquer quantidade de atributos.

Posteriormente são obtidos todos os *Fields* (atributos) para a *feature* selecionada e para cada um deles é criado um *TextView* para exibir o seu nome e um *EditText* para o utilizador inserir o seu valor. Estes são inseridos nos *HashMaps* previamente criados e no final é exibida uma caixa de diálogo do tipo *AlertDialog* contendo um *layout* em que estão dispostos os nomes de todos os atributos e uma caixa de texto para cada um onde o utilizador poderá inserir os seus valores.

Assim que o utilizador insira os valores desejados e pressionar o botão de confirmação, é criado um objeto do tipo *Map<String, Object>* que irá conter os atributos que o utilizador editou. Este objeto corresponde a um mapa *key-value pair* que tem como *keys* os nomes dos atributos editados pelo utilizador e como *values* os seus valores.

Finalmente é chamada a função *updateFeature()* que recebe como parâmetros a *feature* a ser editada, o nome da *FeatureTable* da *geodatabase* em que pertence a *feature,* e os atributos a serem inseridos na *feature*, e que permite inserir os atributos editados pelo utilizador na *feature* selecionada recorrendo à função *updateFeatureAsync()*.

Na Figura 29 está representado o funcionamento desta funcionalidade.

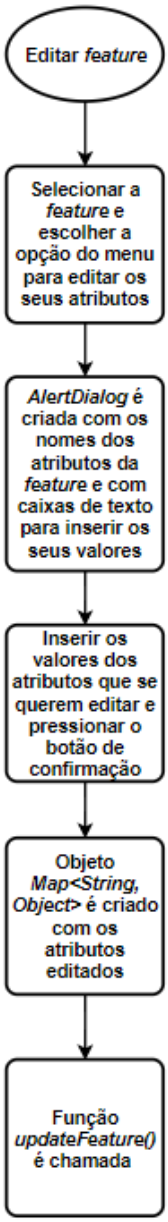

Figura 29 - Fluxograma representativo da opção de editar os atributos de uma feature

Na Figura 30 está representada a janela com os campos para se editar os valores dos atributos da *feature*. Todos os campos são facultativos podendo-se não preencher nenhum, só alguns ou todos.

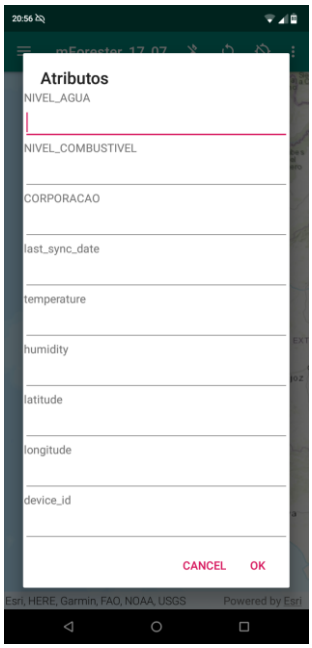

Figura 30 - Janela para editar os valores dos atributos da feature

- **Funcionalidade eliminar *feature***

À semelhança de outras funcionalidades, a que permite eliminar uma *feature* recorre igualmente à função *onSingleTapConfirmed()*.

Quando se seleciona a *feature* que se pretende eliminar é exibido um *Callout* com uma série de opções disponíveis.

Caso a opção de eliminar a *feature* seja selecionada, surge uma caixa de diálogo do tipo *AlertDialog* que pede a confirmação do utilizador.

Se o utilizador não confirmar a eliminação da *feature* a janela é eliminada e a *feature* mantém-se intacta, caso contrário, é chamada a função *deleteFeature().*

A função *deleteFeature()* recebe como parâmetros a *feature* a ser eliminada e a *FeatureTable* em que esta se encontra na *geodatabase.* Nesta função é primeiro verificado se é possível eliminar-se a *feature* e no caso de se poder, procede-se à eliminação da mesma através da função *deleteFeatureAsync()* cuja execução é assíncrona. No caso desta operação ser bem sucedida, o utilizador é notificado do seu sucesso.

Na Figura 31 encontra-se representado o funcionamento desta funcionalidade.

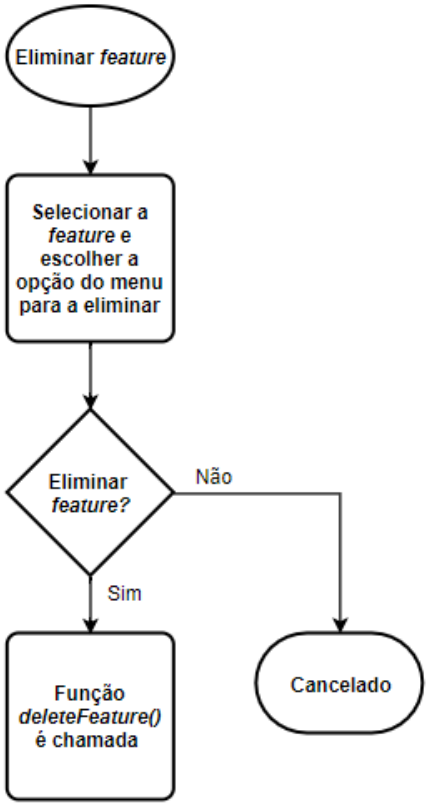

Figura 31 - Fluxograma representativo da opção de eliminar uma feature

- Funcionalidade mover *feature*

A funcionalidade de mover uma *feature* no ecrã também faz uso das capacidades de interação do *MapView* em particular da função *onSingleTapConfirmed()*.

Quando a *feature* a ser movida é pressionada, a função *onSingleTapConfirmed()* é chamada. De seguida é mostrado um *Callout* no ecrã com um conjunto de operações possíveis de se fazer na *feature*. É então selecionada a opção para mover que ativa a *flag* que indica o ínicio da operação de mover uma *feature*.

Quando o utilizador pressionar outro ponto no mapa será de novo chamada a função *onSingleTapConfirmed()* onde será feita a verificação da *flag* e estando esta ativa é chamada a função *moveFeature()*.

A função *moveFeature()* recebe como parâmetros a *feature* a ser movida e o objeto *Point* que tem a nova geometria a ser aplicada. Aqui é atualizada a geometria que fará com que a *feature* no *MapView* seja deslocada para a nova localização. Finalmente é atualizada na sua *FeatureTable* da *geodatabase*.

O fluxograma da Figura 32 representa o funcionamento desta opção.

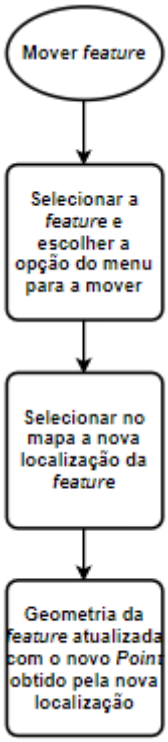

Figura 32 - Fluxograma representativo da opção de mover uma feature

- **Funcionalidade abrir anexos da *feature***

A funcionalidade de abrir anexos também faz uso da *callback onSingleTapConfirmed().*

Quando a *feature* cujos anexos se quer consultar é pressionada, é exibido um *Callout* na interface de utilizador com uma série de opções disponíveis.

Caso a última opção (correspondente aos anexos) seja selecionada, é exibida uma caixa de diálogo do tipo *AlertDialog* que contém um menu de opções possíveis de se fazer com os anexos.

Se a opção selecionada for a de "Ver anexos", é chamada a função *listAttachments()* com o parâmetro "open" que ativará a visualização de anexos. Na função *listAttachments()* é criado um objeto *ListenableFuture<List<Attachment>>* que irá carregar os anexos da *feature* selecionada. Este objeto implementa o padrão de design *Loadable* e assim que os seus metadados sejam carregados, a função *addDoneListener()* é chamada. Nesta função são obtidos os anexos através da função *get(),* os quais são guardados numa variável do tipo *List<Attachment>,* e de seguida é exibida uma caixa de diálogo do tipo *AlertDialog* contendo a sua listagem.

Caso o utilizador selecione algum dos anexos da lista, é chamada a função *fetchAndOpenAttachment(),* dado que o parâmetro passado para a função *listAttachments()* indica a opção de visualização. Nesta função é criado um objeto *ListenableFuture<InputStream>* que irá carregar os dados do anexo selecionado pelo utilizador. Este objeto implementa o padrão de design *Loadable* e quando os seus metadados forem carregados, a função *addDoneListener()* é chamada.

Na função *addDoneListener()* é executada a compressão da imagem em anexo recorrendo a uma *AsyncTask* denominada *BitmapCompressionTask.* Uma *AsyncTask* é uma classe que permite realizar operações de curta duração em *background* e publicar os seus resultados na interface de utilizador, dispensando o utilizador da manipulação de *Threads* ou *Handlers*.

A compressão de imagens é um processo que pode ser demorado e computacionalmente exigente pelo que a sua execução deverá ocorrer numa *thread* separada dado que se ocorrer na *thread* da interface de utilizador poderá bloquear a mesma conduzindo a uma experiência de utilizador desagradável. Desta forma a compressão da imagem é executada em *background* e a sua execução é comunicada ao utilizador através duma *ProgressDialog* que irá ser exibida enquanto a compressão estiver a ocorrer.

Quando esta operação estiver concluída, é criada uma *Intent* para a abrir no visualizador de imagens padrão do dispositivo.

Na Figura 33 está representado um fluxograma do funcionamento desta opção.

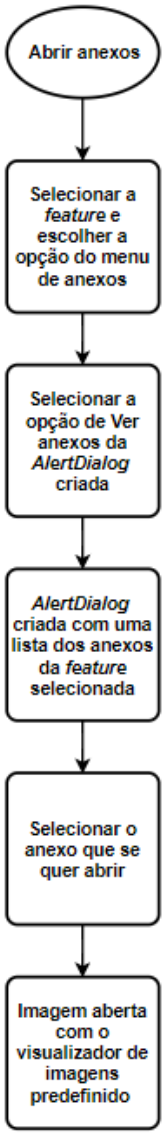

Figura 33 -Fluxograma representativo da opção de abrir anexos

Na Figura 34 está representada a janela com uma listagem dos anexos existentes na *feature* selecionada

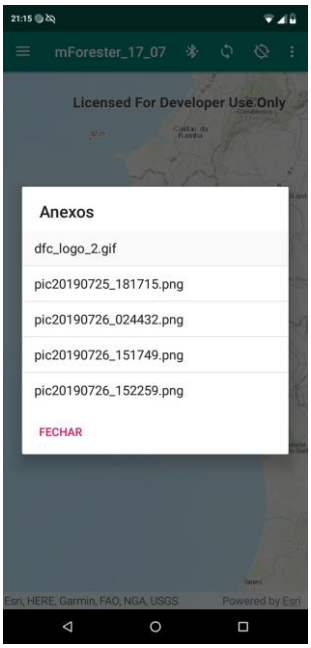

Figura 34 - Janela com a lista de anexos duma feature selecionada

- Funcionalidade adicionar anexos à *feature*

A funcionalidade de adicionar anexos faz igualmente uso da função *onSingleTapConfirmed()*.

Quando é selecionada a *feature* à qual se quer adicionar anexos é exibido um *Callout* com um conjunto de opções disponíveis.

Se a última opção (correspondente aos anexos) for selecionada, é exibida uma caixa de diálogo do tipo *AlertDialog* que contém um menu de opções possíveis de se fazer com os anexos.

Caso a opção escolhida seja a de "Adicionar anexos", é chamada a função *addAttachment()* que exibe uma *AlertDialog* em que o utilizador escolhe se pretende que os anexos venham da câmera ou da galeria, chamando posteriormente a *callback attachmentCallback()* onde será gerida a opção.

No caso da escolha ser a opção da câmera, é criado um ficheiro que receberá a fotografia a ser tirada. Posteriormente é definido um *Intent* que irá iniciar a aplicação da câmera do dispositivo através da função *startActivityForResult()*.

Esta função tem como parâmetros o *Intent* para iniciar a aplicação da câmera e um *id* denominado *request code* que permitirá identificar o pedido quando a função *onActivityResult()* for chamada.

Depois da fotografia ser tirada e voltar-se à aplicação, é chamada automaticamente a função *onActivityResult()* onde será então verificado o *request code* correspondente ao *Intent* da aplicação da câmera. De seguida é declarado um objeto *File* a partir da localização do ficheiro da fotografia e outro objeto do tipo *Bitmap* que receberá a fotografia proveniente da aplicação da câmera.

Tendo a fotografia guardada, é exibida uma *AlertDialog* que pergunta ao utilizador o estado da ocorrência que será posteriormente sobreposto sobre a fotografia tirada recorrendo à função *processBitmap()*. Este processo de sobreposição destes dados em conjunto com a *timestamp* e a localização, será explicado na secção 3.4.2.3.

Quando a sobreposição dos dados estiver concluída é executada a compressão da imagem recorrendo à *AsyncTask BitmapProcessingTask.* No final do processo de compressão, a nova imagem com os dados sobrepostos é convertida para um *byte array* através da função *readFileToByteArray()* e de seguida adicionada como anexo à *feature* selecionada recorrendo à função *addAttachmentAsync()*. Esta função recorre ao padrão de design *Loadable* e assim que esteja concluída e seja chamada a função *addDoneListener()*, procede-se à atualização da *feature* com o novo anexo na sua *FeatureTable* da *geodatabase* recorrendo à função *updateFeatureAsync().*

Por outro lado, se se pretender adicionar um anexo a partir da galeria, é definido um *Intent* que permitirá abrir a aplicação de visualização de imagens. Tal é possível com a função *startActivityForResult()* em que é passado o *request code* que irá identificar este *Intent*.

Na função *onActivityResult()* é verificado o *request code* e, posteriormente, é obtido o *URI* da imagem recebida da galeria. Um *URI (Universal Resource Identifier)* é uma sequência de caracteres que permite identificar um recurso. Este é usado para obter a localização do ficheiro de forma a poder declarar um objeto *File*. A partir do objeto *File* é criado um *byte array* através da função

readFileToByteArray() que será adicionado como anexo à *feature* selecionada graças à função *addAttachmentAsync().* No final, à semelhança do que foi feito para a imagem proveniente da câmera, a *feature* é atualizada na sua *FeatureTable* da *geodatabase* recorrendo à função *updateFeatureAsync()* que recorre ao padrão de design *Loadable* chamando a função *addDoneListener()* quando estiver concluída. Nesta função o utilizador será informado quanto ao sucesso da operação.

A Figura 35 contém um fluxograma do funcionamento desta funcionalidade.

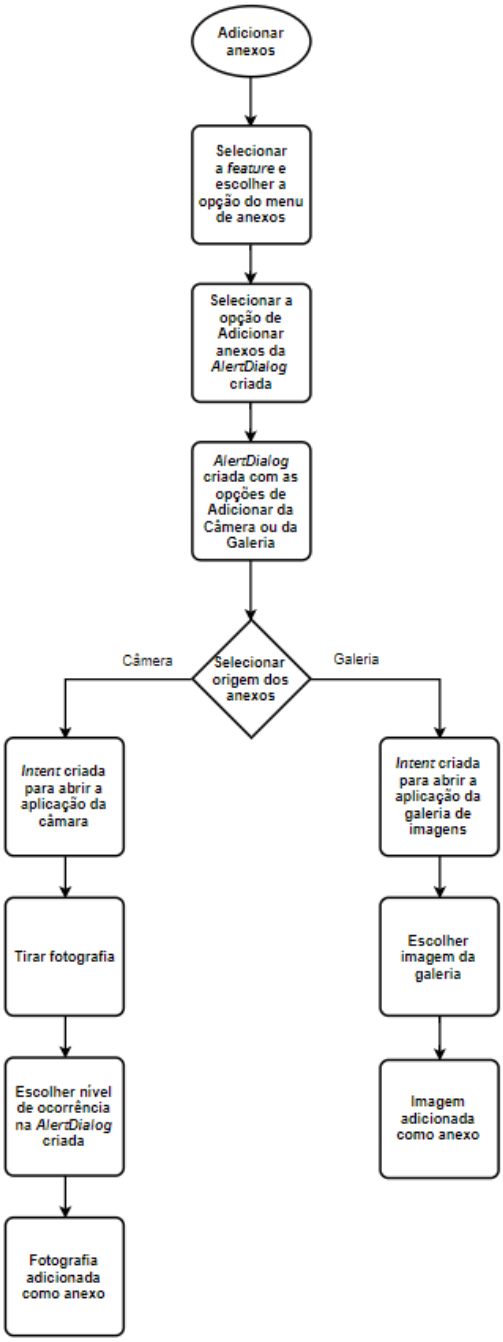

Figura 35 - Fluxograma representativo da opção de adicionar anexos

Na Figura 36 está representa uma janela que pergunta ao utilizador se pretende que os anexos venham da câmera ou da galeria.

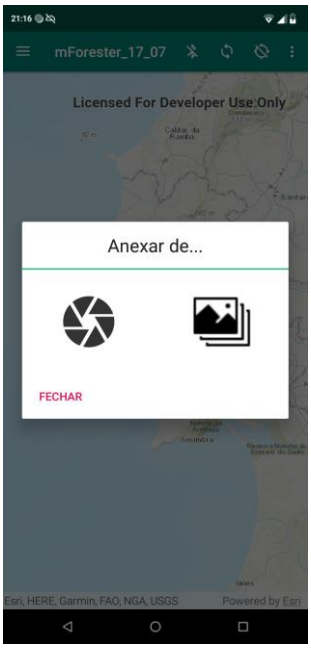

Figura 36 - Janela que pergunta ao utilizador de onde pretende adicionar um anexo

- **Funcionalidade eliminar anexos da *feature***

A funcionalidade de adicionar anexos faz igualmente uso da função *onSingleTapConfirmed()*.

Quando se seleciona uma *feature* a que se queira eliminar um anexo, um *Callout* é exibido com uma série de opções disponíveis.

Se a última opção (correspondente aos anexos) for pressionada, é exibida uma caixa de diálogo do tipo *AlertDialog* que contém um menu de opções possíveis de se fazer com os anexos.

Caso a opção escolhida seja a de "Eliminar anexos", é então chamada a função *listAttachment()* com o parâmetro "delete" que ativará a eliminação de anexos. Na função *listAttachments()*, tal como foi explicado anteriormente, é obtida uma lista de anexos da *feature* selecionado e, posteriormente, é exibida uma caixa de diálogo do tipo *AlertDialog* contendo essa lista. Dado que se passou o parâmetro "delete", após se ter selecionado um dos anexos é exibida uma *AlertDialog* que pede a confirmação do utilizador para eliminar o anexo selecionado. Se o utilizador conceder a autorização, é chamada a função *deleteAttachment()*.

A função *deleteAttachment()* recebe como parâmetro o índice do anexo selecionado pelo utilizador na lista de anexos exibida anteriormente. Este índice permite aceder ao anexo no objeto *List<Attachment>* que se criou na função *listAttachments()* assim como à lista que contém os nomes dos anexos que foram exibidos na *AlertDialog* anterior.

De seguida é criado um objeto do tipo *ListenableFuture<Void>* e executada a função *deleteAttachmentAsync()* para eliminar o anexo selecionado da *feature*. Este objeto recorre ao padrão de design *Loadable* e assim que a eliminação do anexo esteja concluída, a função *addDoneListener()* é executada.

Finalmente, nesta função é atualizada a *feature* à qual se eliminou o anexo na sua *FeatureTable* da *geodatabase* através da função *updateFeatureAsync()*. Assim que esta operação esteja concluída e a função *addDoneListener()* seja executada, o utilizador é informado do sucesso da mesma.

Na Figura 37 está representado um fluxograma do funcionamento desta opção.

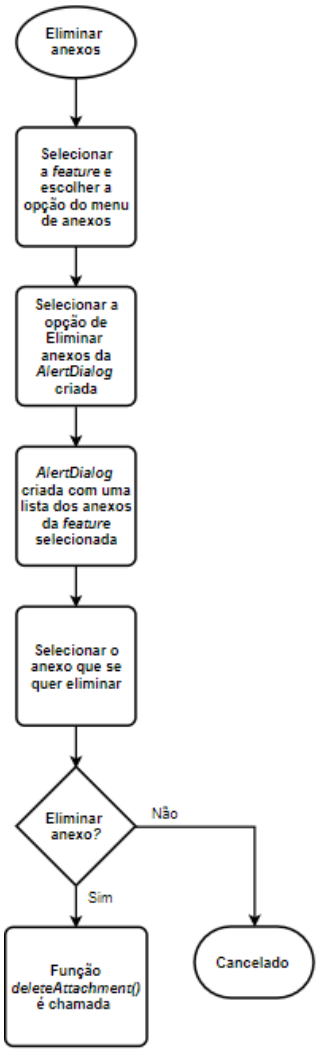

Figura 37 - Fluxograma representativo da opção de eliminar anexos

3.4.2.3. Implementação da componente foto c/ *overlay*

A implementação da componente de foto com *overlay* de informação foi realizada de forma integrada com a funcionalidade de adicionar anexos provenientes da câmera.

O objetivo desta componente é o de sobrepor uma camada na fotografia que contenha dados de interesse como uma *timestamp* de quando foi captada a fotografia, a sua localização geográfica e a informação do estado da ocorrência no local.

A Figura 38 apresenta um diagrama do funcionamento desta componente.

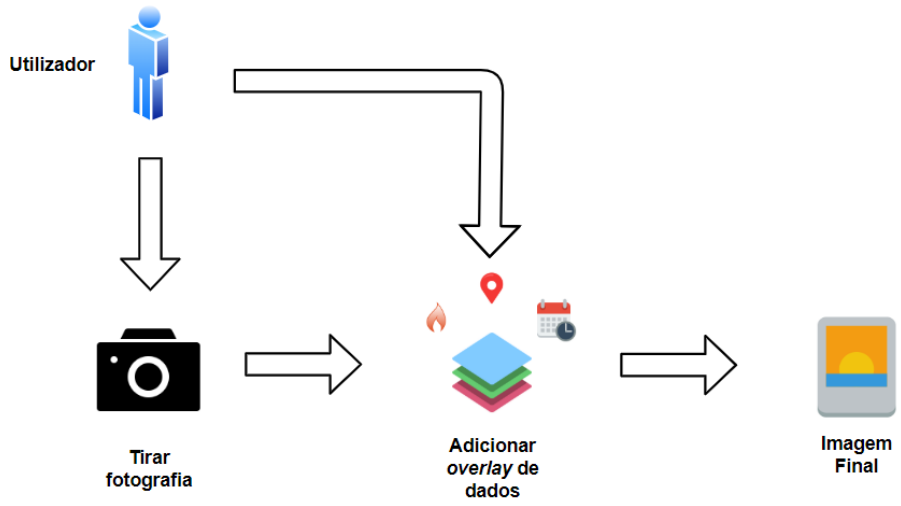

Figura 38 - Diagrama representativo do funcionamento da componente de fotografia com overlay

A funcionalidade inerente a esta componente está encapsulada na função *processBitmap()*.

Esta função recebe como parâmetros de entrada um objeto *Bitmap*, que corresponde à imagem original à qual se vai sobrepor os dados, e uma *string* contendo o estado da ocorrência escolhido pelo utilizador.

No início é criado um novo *Bitmap* com as dimensões do original, que terá a imagem resultante do processamento. De seguida é criado um objeto do tipo *Canvas* onde serão chamadas as funções de desenho. Este objeto recebe como parâmetro o *Bitmap* final onde serão incluídos os dados de interesse. É depois chamada a função *drawBitmap()* para que, na prática, seja feita uma cópia do *Bitmap* original para o final.

Posteriormente, é criado um objeto do tipo *Paint* onde serão configurados parâmetros como o tamanho do texto, a sua cor e o seu estilo. Este objeto permite configurar como serão representados os dados sobrepostos na fotografia.

É então obtida a *timestamp* em que foi captada a foto através dos objetos *Calendar* e *SimpleDateFormat* e é criada uma *string* que contém a localização formatada de uma forma legível.

De seguida, é criado um objeto do tipo *Rect* que permitirá obter o tamanho do texto a ser sobreposto recorrendo à função *getTextBounds(),* que é chamada de seguida. Com o tamanho do texto obtido, é então chamada a função *drawText()* do objeto *Canvas* que irá sobrepor o texto sobre a fotografia. Esta função aceita como parâmetros uma *string* com os dados a serem sobrepostos, as coordenadas *x* e *y* onde sobrepor e o objeto *Paint* com as caraterísticas do texto.

Este processo de obtenção do tamanho do texto e sua sobreposição é repetido para cada um dos três dados a incluir na fotografia: *timestamp*, localização geográfica e estado da ocorrência.

No final será executada a *AsyncTask* denominada *BitmapProcessingTask* que será responsável por comprimir a fotografia processada no formato *PNG*.

Esta componente permite assim sobrepor dados sobre uma fotografia sendo facilmente adaptada para qualquer tipo de dados desde que se tenha o cuidado de posicionar corretamente os mesmos na fotografia. Deste modo, se se pretender inserir mais alguma informação, as alterações só precisam de ser realizadas na função *processBitmap()*.

Na Figura 39 está representado o resultado final duma imagem à qual foi sobreposta um *timestamp*, localização geográfica e nível da ocorrência previamente selecionada pelo utilizador.

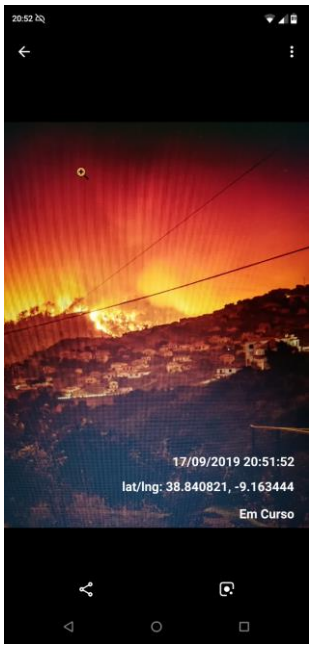

Figura 39 - Imagem final depois de ser sobreposta a camada de informação

3.4.2.4. Implementação da componente *BLE*

O desenvolvimento da componente responsável pela comunicação com o *device* através da interface Bluetooth BLE , divide-se em três partes:

- *DeviceScan* – esta *Activity* é responsável por procurar dispositivos *BLE* e retorna para a *Activity* principal (*MainActivity*) o nome e o endereço do dispositivo selecionado da lista pelo utilizador.
- *BluetoothLeService* – serviço responsável por gerir a conexão com o dispositivo *BLE* e por receber os dados enviados por ele.
- *BroadcastReceiver* – componente que recebe as notificações (*broadcasts*) provenientes do serviço *BluetoothLeService* com as alterações do estado da conexão e os novos dados sensoriais.

Cada uma destas componentes interliga-se com as outras de forma a atingir o objetivo final de extrair os dados sensoriais do *device* através da interface de comunicação *BLE*.

De seguida será descrito o processo de implementação de cada uma das partes que compõe a componente *BLE.*

Na Figura 40 está representado um diagrama elucidativo do funcionamento deste componente da aplicação.

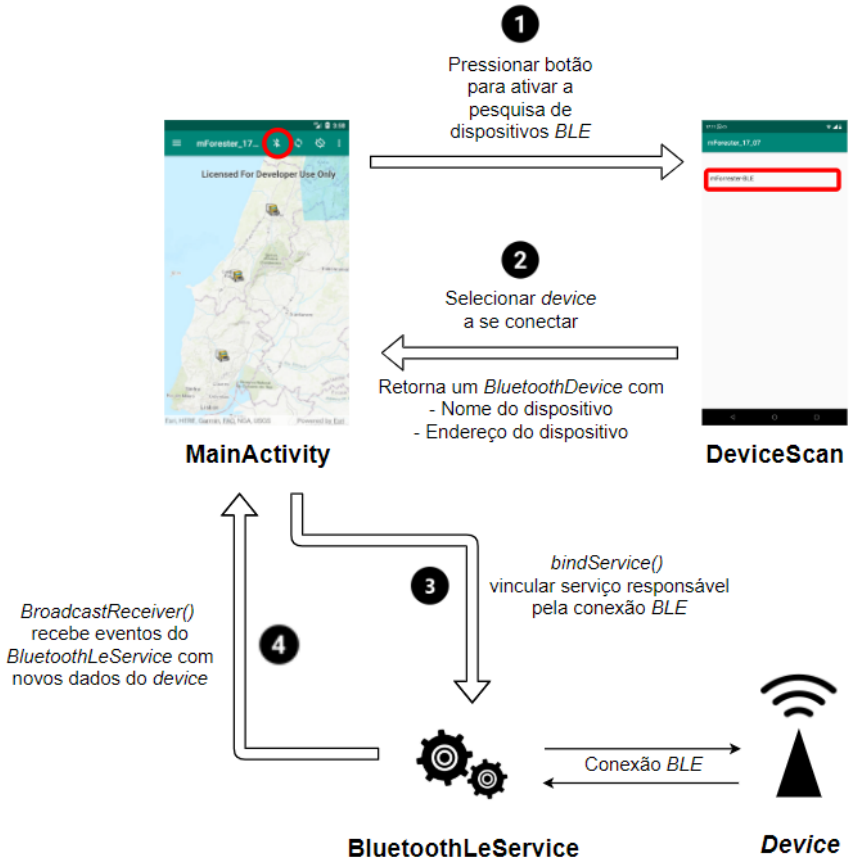

Figura 40 - Diagrama representativo do funcionamento geral da componente BLE da aplicação

- Desenvolvimento da *Activity DeviceScan*

A componente de *BLE* da aplicação engloba três etapas importantes: procura de dispositivos *BLE*, estabelecimento da conexão com o dispositivo e leitura dos dados por ele transmitidos.

A *Activity DeviceScan* é responsável pela procura de dispositivos *BLE* na proximidade do *tablet* a que se possa conectar.

A inicialização desta *Activity* ocorre quando o utilizador pressiona o botão com o símbolo do *bluetooth* presente na *ActionBar* (barra de ferramentas presente no topo da *Activity*). Com a sua criação, é chamada a função *onCreate()*

onde são inicializadas as variáveis utilizadas e configurado o *BLE* através da função *setupBluetooth().*

Nesta função, é inicialmente verificado se o *tablet* dispõe de *Bluetooth BLE*, terminando a aplicação caso este não possua. No caso de possuir, é criado um objeto do tipo *BluetoothManager* do qual se obtém um objeto *BluetoothAdapter*. O *BluetoothAdapter* é um objeto que representa o hardware do *Bluetooth* no *tablet*. Com este, é possível realizar operações como procurar dispositivos *BLE*, obter os dispositivos emparelhados com o *tablet*, etc.

De seguida é iniciada a procura de dispositivos *BLE*, o que é feito na função *startScanning(),* que ocorre por um período de 5 segundos após o qual é chamada a função *stopScanning()* que termina a procura.

Durante este processo são reportados os novos dispositivos encontrados para uma *callback* definida por um objeto do tipo *ScanCallback*. No construtor deste objeto implementa-se a função *onScanFailed()* que permite alertar o utilizador da falha na procura de dispositivos, a função *onScanResult()* e a função *onBatchScanResults()* que chamam a função *processResult()* para onde enviam como argumento o resultado da procura.

Deste resultado é extraído o dispositivo *Bluetooth,* representado por um objeto *BluetoothDevice,* que é adicionado a uma lista que contém os dispositivos encontrados e cujo nome é adicionado à *ListView* na interface de utilizador.

Após os resultados terem sido processados, o utilizador pode selecionar qual o dispositivo a que se quer conectar. Quando a escolha do dispositivo for efetuada, é criado um objeto *Intent* para o qual se passa o dispositivo escolhido e a *Activity DeviceScan* é terminada voltando-se assim à *Activity MainActivity.*

Já na *Activity MainActivity* é executada a função *onActivityResult()* onde é verificado o *request code* da *Intent* que se criou para ir para a *Activity DeviceScan*.

De seguida é obtido o nome e endereço do dispositivo e chamada a função *bindService()* que permite vincular um serviço, com uma *Intent* para a classe *BluetoothLeService* e com um objeto *ServiceConnection* que monitoriza o estado do serviço.

Desta forma vincula-se o serviço *BluetoothLeService* que a partir desse momento fica responsável por estabelecer a conexão com o dispositivo *BLE* e receber os dados por ele enviados.

A Figura 41 exibe um diagrama elucidativo do funcionamento da procura de dispositivos *BLE*.

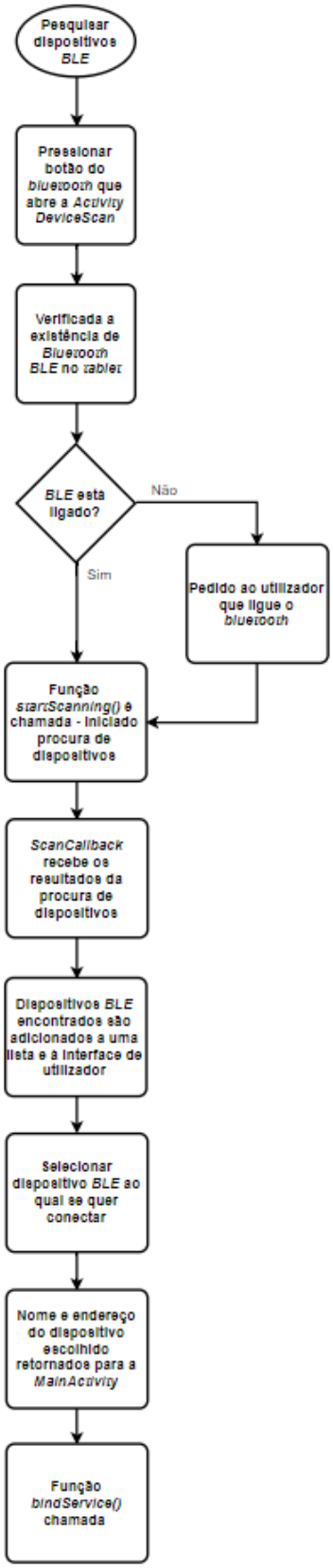

Figura 41 - Diagrama elucidativo do funcionamento da pesquisa de dispositivos BLE

- Desenvolvimento do serviço *BluetoothLeService*

A implementação do serviço *BluetoothLeService* começa a partir do momento em que este se vincula à *Activity MainActivity*.

A razão pela qual se optou por implementar a conexão *Bluetooth BLE* com recurso a um serviço foi pelo facto desta conexão não necessitar de uma interface de utilizador e poder ser corrida em segundo plano.

Além disso, os serviços que sejam vinculados disponibilizam uma interface *client-server* que permite que outros componentes da aplicação interajam com o serviço para troca de dados. Deste modo é possível usar componentes como *BroadcastReceivers* que subscrevem aos dados provenientes do serviço permitindo assim comunicar entre processos.

Após o *bindService()* ser chamado, com uma *Intent* para a classe *BluetoothLeService* e um objeto *ServiceConnection* que monitorizará o serviço, é chamada a função *onBind()* no serviço *BluetoothLeScanner.* Esta função retorna um objeto *IBinder,* que servirá como canal de comunicação para o serviço, para a função *onServiceConnected()* que é implementada na declaração do objeto *ServiceConnection* na *MainActivity*. Para além desta função, também a *onServiceDisconnected()* é implementada para lidar com a desconexão do serviço.

Na função *onServiceConnected()*, é obtido através do objeto *IBinder* recebido na função, um objeto *BluetoothLeService* que permitirá aceder aos métodos implementados na classe *BluetoothLeService*.

De seguida é chamada a função *initialize()* do objeto *BluetoothLeService* onde é criado um objeto do tipo *BluetoothManager* a partir do qual se obtém um objeto do tipo *BluetoothAdapter* que fará a interface com o rádio *Bluetooth*.

Posteriormente é chamada a função *connect()* do objeto *BluetoothLeService* que recebe como parâmetro o endereço do dispositivo *BLE* previamente selecionado na *Activity DeviceScan*. Aqui é obtido o *BluetoothDevice* através da função *getRemoteDevice()* do *BluetoothAdapter* que permite obter um dispositivo *BLE* detetado pelo rádio *Bluetooth* através do seu endereço.

Neste objeto *BluetoothDevice* é chamada a função *connectGatt()* que representa uma interface para a funcionalidade da especificação *Bluetooth GATT* permitindo assim comunicar com dispositivos *BLE*. A esta função é passada um *callback* do tipo *BluetoothGattCallback*, onde são implementados os métodos *onConnectionStateChange()* e *onServicesDiscovered()*. Esta função retorna um objeto do tipo *BluetoothGatt* onde poderão ser efetuadas as operações de cliente no *GATT*.

Depois da chamada da função *connectGatt(),* o *tablet* conecta-se ao *GATT Server* presente no dispositivo *BLE* selecionado e a função *onConnectionStateChange()* é chamada indicando um evento de conexão no *GATT Server*. Este evento é reportado ao *BroadcastReceiver* presente na *MainActivity* através da função *broadcastUpdate()*.

Após ser estabelecida a conexão entre *tablet* e *GATT Server* (presente no dispositivo *BLE*), é iniciada a procura de *Services* disponibilizados pelo dispositivo através da função *discoverServices()* do objeto *BluetoothGatt*. Com a chamada desta função é despoletada a *callback onServicesDiscovered()*.

Nesta *callback* começa-se por reportar ao *BroadcastReceiver* um evento de descoberta de *Services*. Posteriormente são obtidos todos os *Services* (representados por um *List de* objetos *BluetoothGattService*) presentes no dispositivo através da função *getServices()* do objeto *BluetoothGatt* recebido na *callback,* e verificada a existências dos *Services* definidos na estrutura de dados do *device* (ver secção 3.3.2.2.).

Caso estes existam, são obtidas as suas *Characteristics* através da função *getCharacteristics()* de cada *BluetoothGattService que* são adicionadas a uma *ArrayList.*

No final desta *callback,* é chamada a função *enableCharacteristicNotification()* que tem como argumento o *BluetoothGatt* recebido nesta função.

Na função *enableCharacteristicNotification()* são ativadas as notificações para as *Characteristics* respeitantes aos dados sensoriais que se quer consumir. Tal é feito em dois passos: primeiro obtém-se uma *Characteristic* da *ArrayList* que as contém e chama-se a função *setCharacteristicNotification()* do objeto

BluetoothGatt enviando como parâmetros a *Characteristic* e o valor *booleano true* que permite registar localmente no *tablet* a ativação das notificações para esta *Characteristic*, em segundo lugar obtém-se o *Client Characteristic Configuration Descriptor* da *Characteristic* e altera-se o seu valor de forma a ativar as notificações chamando, posteriormente, a função *writeDescriptor()* do *BluetoothGatt* para que seja registada essa alteração do dispositivo *BLE.*

Após a operação de escrita do *Descriptor* no dispositivo *BLE* ter sido efetuada, é chamada a *callback onDescriptorWrite().* Nesta *callback* é removida da *ArrayList* a última *Characteristic* cuja notificação tinha sido ativada e chamada novamente a função *enableCharacteristicNotification().* Este processo irá repetir-se até que a *ArrayList* que contém as *Characteristics* a que se quer subscrever estiver vazia. Tal significa que as notificações foram ativadas para todas as *Characteristics* de interesse.

A partir deste momento, cada vez que o *device* notificar novos valores, será chamada a *callback onCharacteristicChanged().* Nesta *callback* é chamada a função *broadcastUpdate()* enviando a indicação de novos dados e a *Characteristic* a que estes correspondem. Na função *broadcastUpdate()* são obtidos os valores da *Chraracteristic* recebida e enviados para a *MainActivity* através da função *sendBroadcast().*

Finalmente, na *MainActivity* são recebidos os valores através do componente *BroadcastReceiver,* cujo funcionamento será explicado de seguida.

A Figura 42 exibe um diagrama elucidativo da implementação do serviço *BluetoothLeService*.


Estabelecer conexão com dispositivo
bindService() vincula serviço BluetoothLeService à MainActivity
Callback onServiceConnected() chamada
BLE inicializado
Conexão estabelecida com o device
onConnectionStateChange() chamada com o evento de ligação estabelecida
Função discoverServices() chamada

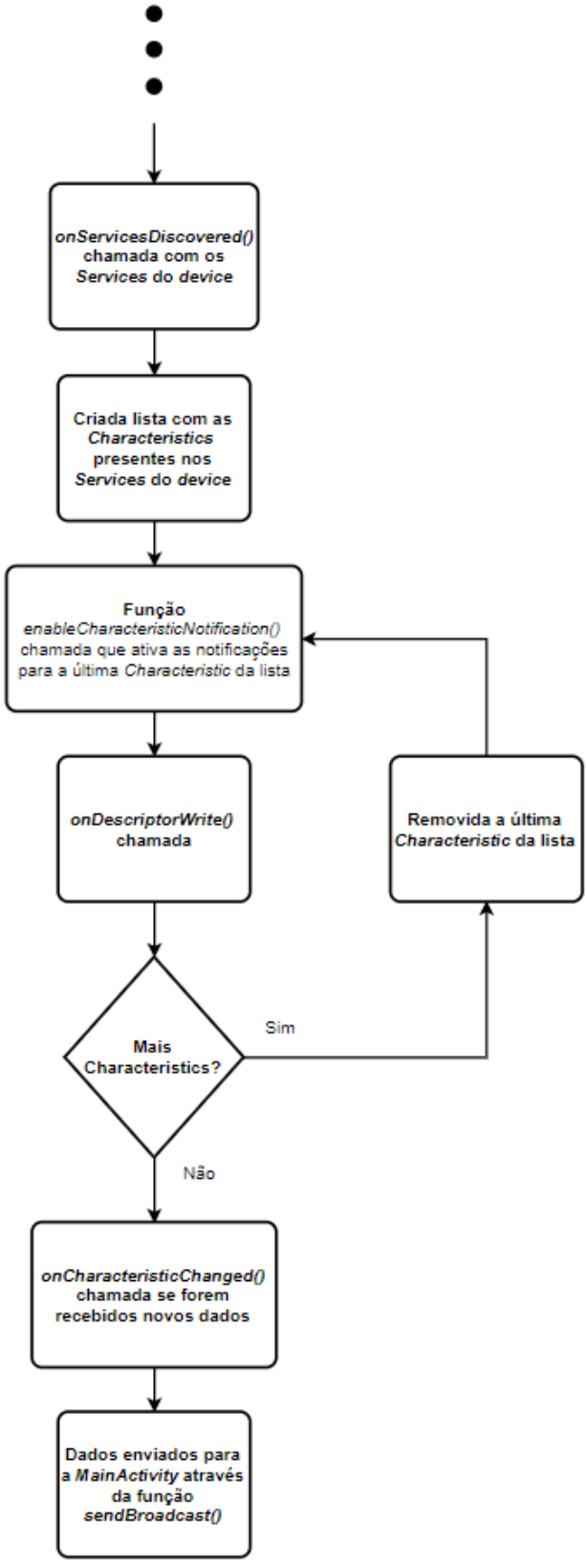

Figura 42 - Diagrama elucidativo do funcionamento da pesquisa de dispositivos BLE

- **Desenvolvimento da componente *BroadcastReceiver***

Com a procura de dispositivos *BLE* e gestão da conexão implementados, resta assim desenvolver o mecanismo de comunicação dos dados para outras componentes da aplicação para que estes sejam consumidos.

Como explicado anteriormente, os serviços dispobilizam uma interface *client-server* que permite a comunicação inter-processos podendo-se transferir dados entre várias componentes da aplicação.

De forma a poder receber os dados enviados pelo *BluetoothLeService* foi criado um *BroadcastReceiver* na *MainActivity.*

É necessário registar o *BroadcastReceiver,* o que é feito através da função *registerReceiver()* que é chamada na função *onResume().* A função *registerReceiver()* recebe como parâmetros o objeto *BroadcastReceiver* e um filtro do tipo *IntentFilter* que terá as *broadcast Intents* a que se quer subscrever. Por outro lado, é também necessário remover o registo do *BroadcastReceiver* quando o utilizador não está a interargir com a *MainActivity*, o que é realizado pela função *unregisterReceiver()* na função *onPause()*.

Neste *BroadcastReceiver* é implementada a função *onReceive()* que é chamada quando são as recebidas *broadcast Intents* que correspondam a alguma das presentes no *IntentFilter*.

Quando é recebida uma *broadcast Intent* que corresponda a alguma presente no filtro, é verificado qual o tipo de *broadcast Intent* na função *onReceive().* Existem quatro tipos de *broadcast Intents* no filtro: *BluetoothLeService.ACTION_GATT_CONNECTED*, *BluetoothLeService.ACTION_GATT_DISCONNECTED*, *BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED* e *BluetoothLeService.ACTION_DATA_AVAILABLE*.

Os dois primeiros tipos de *broadcast Intents* advertem para eventos de conexão e desconexão, respetivamente, do *tablet* com o *GATT Server* (dispositivo *BLE*) sendo que o utilizador é notificado através de uma *Toast message* desses eventos.

O terceiro tipo de *broadcast Intent* adverte para o evento de leitura dos *Services* disponibilizados pelo dispositivo *BLE,* fazendo depois uma listagem desses *Services*.

Finalmente, o quarto tipo de *broadcast Intent* adverte para a existência de novos dados provenientes do dispositivo *BLE.* Com este evento, são então verificados os valores que vêm na *Intent* e atualizados os que forem detetados. Neste ponto há uma estreita integração entre as componentes *SIG* e *BLE.*

De seguida, é criado um objeto do tipo *Map<String, Object>* que conterá os atributos a serem passados à *feature* representativa do dispositivo *BLE* no mapa.

A este objeto serão passados os dados sensoriais sendo que as coordenadas são previamente verificadas para que, em caso de falha na obtenção da posição, a localização da *feature* não seja alterada para um local incorreto e assim permaneça a última posição válida conhecida.

Finalmente, é chamada a função *searchForDevice()* onde é verificada a existência da *feature* representativa do dispositivo *BLE* na respetiva *FeatureTable* da *geodatabase.*

Caso esta exista, são atualizados os seus atributos e a sua localização. Caso contrário, é criada uma *feature* com os dados recebidos do dispositivo *BLE* como atributos, na localização em que este se encontra.

A Figura 43 exibe um diagrama elucidativo da implementação da fase de recepção e consumo dos dados do dispositivo *BLE* recorrendo a um *BroadcastReceiver.*

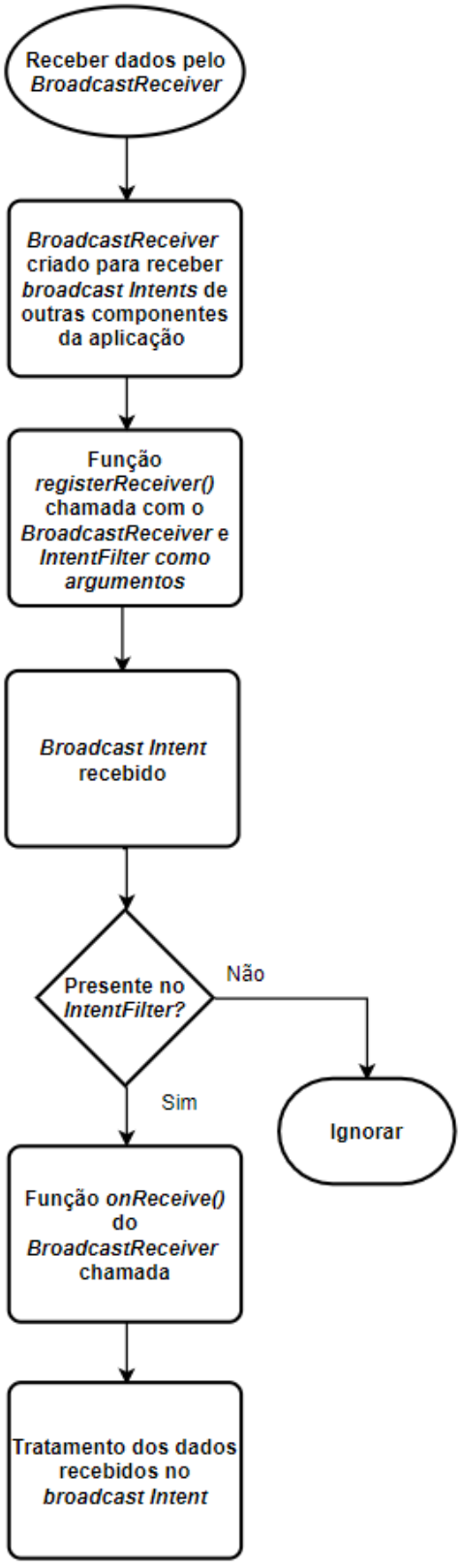

Figura 43 - Diagrama elucidativo da implementação da receção e tratamento de dados através de um BroadcastReceiver

4.5. Implementação do *Feature Service* no *ArcGIS Online*

A implementação do *Feature Service* foi realizada com recurso ao *ArcGIS Online*. Para tal, foi criada uma nova *Feature Layer* na página *Content* da conta pessoal e partiu-se de um *template* que permitia adicionar três *layers* do tipo *Point*, *Polyline* e *Polygon*.

De seguida, é definida a extensão do mapa para a *Feature Layer* e depois é atribuído o seu título, *tags* que ajudam a identificar o mapa e um sumário (opcional) do que esta diz respeito.

Finalmente é então aberta a *Feature Layer* no *Map Viewer* onde se pode editar as suas *features* e seus atributos.

A Figura 44 mostra a *Feature Layer* aberta no *Map Viewer*.

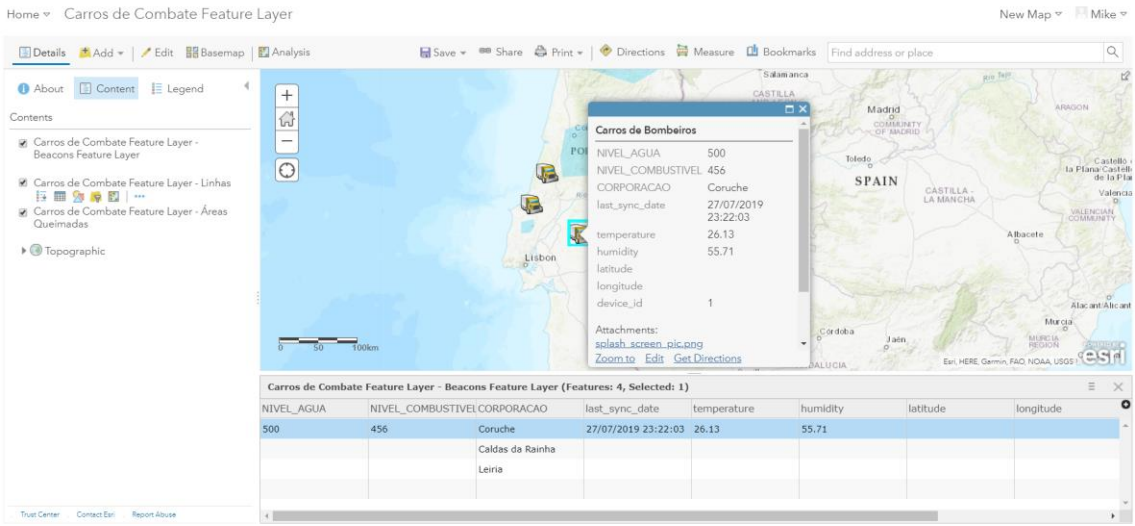

Figura 44 - *Feature Layer* criada aberta no *Map Viewer* onde pode ser editada

No final de se editar a *Feature Layer,* fica disponibilizado no *ArcGIS Online* um *Feature Service* a partir do qual se pode sincronizar uma *geodatabase* presente no *tablet*.

4.5.1. Arquitetura das *Feature Layers* no *ArcGIS Online*

Como referido anteriormente, existem três *layers* de tipos diferentes nesta *Feature Layer*, sendo que a cada uma foi atribuída uma finalidade específica.

Além disso, cada uma destas *layers* contém uma estrutura de dados própria com parâmetros de exemplo. Esta estrutura de dados pode ser facilmente adaptada no *ArcGIS Online* às necessidades do utilizador final sendo que para isso altera-se os *Fields* presentes na tabela de dados associado à *layer.*

A *layer* do tipo *Point* contém como *features* os *devices* instalados nos carros de bombeiros. Na Tabela 8 está representada a estrutura de dados desta *layer* com os *Fields* criados e as suas descrições.

Há que salientar que nesta *layer* em particular, os dados dos seus *Fields* são provenientes do *device* que está conectado ao *tablet* através da interface *Bluetooth BLE*. Estes dados são guardados na *geodatabase* e sincronizados com o *Feature Service* quando o utilizador assim o desejar.

Tabela 8 – Estrutura de dados das *features* do tipo *Point*

Nomes dos *Fields*	Descrição
CORPORACAO	Nome da corporação de bombeiros a que pertence o veículo que contém este *device*
last_sync_date	Data e hora da última sincronização com esta *feature*
temperature	Temperatura do ar medida no local deste *device*
humidity	Humidade relativa do ar medida no local deste *device*
latitude	Latitude onde está situado este *device*
longitude	Longitude onde está situado este *device*

device_id	Identificador único de cada *device* que os permite distinguir dos restantes

A *layer* do tipo *Polyline* permite adicionar linhas ao mapa que possam identificar, por exemplo, uma rota percorrida. Na Tabela 9 está representada a estrutura de dados desta *layer* com o seu único *Field* e a sua descrição.

Tabela 9 - Estrutura de dados das *features* do tipo *Polyline*

Nomes dos *Fields*	Descrição
COMPRIMENTO	Comprimento da *Polyline* criada

A *layer* do tipo *Polygon* permite adicionar polígonos ao mapa que possam identificar, por exemplo, uma área de incidência de um incêndio. Na Tabela 10 está representada a estrutura de dados desta *layer* com o seu único *Field* e a sua descrição.

Tabela 10 - Estrutura de dados das *features* do tipo *Polygon*

Nomes dos *Fields*	Descrição
AREA	Área do *Polygon* criado

Com a estrutura de dados definida para os três tipos de *layers*, o utilizador pode agora manipular as *features* presentes no mapa e os seus *Fields* (atributos).

4

Conclusões e trabalho futuro

Neste capítulo serão apresentadas as conclusões relativas ao projeto desenvolvido nesta dissertação. Para além disso, serão também discutidas algumas limitações do sistema implementado e apresentadas sugestões de melhorias a serem aplicadas em trabalhos futuros.

4.1. Conclusão

O sistema de combate aos fogos rurais em Portugal ainda não conta com um suporte significativo de ferramentas tecnológicas que integrem várias áreas científicas afetas aos incêndios rurais.

Grande parte da tecnologia de suporte ao combate, nomeadamente SIGs e simuladores de comportamento do fogo, está centralizada nos PCO, CDOS e CNOS estando por isso a sua informação dependente de uma comunicação fidedigna com os operacionais no TO.

A comunicação no TO assenta, essencialmente, nos sistemas de VHF/AM (Banda Aeronáutica), de VHF/FM dos quais se incluem as redes REPC e ROB, e a

rede SIRESP havendo também a utilização de redes municipais ou até de redes móveis. As comunicações feitas por estes sistemas são por voz pelo que poderão haver constrangimentos na operacionalidade dos meios no TO pela dificuldade que possa haver em transmitir informações como a localização dos meios no terreno e de pontos de abastecimento de água.

Torna-se por isso necessário haver um sistema de fácil utilização ao alcance dos meios no TO que integre várias áreas de conhecimento científico envolvidas no combate aos incêndios e que se torne uma ferramenta de apoio à decisão para as entidades envolvidas no combate aos incêndios rurais.

De forma a procurar responder a esta necessidade foi proposto um sistema que visa ser o primeiro passo na integração de vários módulos que auxiliem os meios no TO a tomar decisões informadas.

O sistema desenvolvido é constituído por uma aplicação *tablet* para a plataforma *Android* que disponibiliza funcionalidades de um *SIG*, integra dados sensoriais através do desenvolvimento de uma interface *Bluetooth BLE* que comunica com um dispositivo sensorial e fotografias que apresentem dados inerentes ao local fotografado.

A aplicação foi desenvolvida recorrendo a tecnologia *ESRI*, mais concretamente ao *SDK* disponibilizado para desenvolver aplicações *Android*. A sua utilização implica o pagamento de uma licença no entanto, dado que esta permite o acesso a suporte da *ESRI* e que atualmente o Estado português e outras instituições nacionais recorrem a tecnologia desta empresa, poder-se-à justificar o investimento.

Além disso, dado que se utilizou o *SDK*, a interface de utilizador pode ser adaptada de forma a tornar-se o mais intuitiva possível, abstraindo assim o utilizador de toda a funcionalidade subjacente tornando a sua experiência o mais fácil e dinâmica possível.

Em conjunto com a aplicação foi também desenvolvido o dispositivo sensorial (*device*) com o qual esta comunica, que permite agregar um número variável de sensores cujos dados são incorporados numa estrutura de dados pré-

definida e comunicados através da interface *BLE*, podendo também a interface *Wifi* ser usada.

Este sistema foi pensado segundo uma arquitetura modular que permita, futuramente, incluir outros módulos que agreguem imagens de satélite da zona duma ocorrência, incorporem mais funcionalidades típicas de um SIG, comuniquem com um simulador de comportamento de fogo *web-based* ou até que incluam mais sensores no *device* que transmitam dados que alimentem esse simulador.

Pretende-se que no futuro seja uma ferramenta completa de apoio a decisão que auxilie os meios envolvidos no combate aos incêndios rurais a tomar decisões fundamentadas num conjunto alargado de dados inerentes às várias dimensões de um incêndio rural.

4.2. Requisitos implementados

O concepção do sistema passou por várias etapas em que se partiu pelo desenvolvimento do *device* passando-se, posteriormente, para o desenvolvimento da aplicação.

Na Tabela 5 foram apresentados os requisitos que este projeto procurou resolver, tendo-se conseguido realizar a grande maioria deles. Na Tabela 11 são apresentados os requisitos que foram implementados, implementados parcialmente e não implementados.

Tabela 11 – Requisitos implementados, não implementados e parcialmente implementados

Requisitos	Resultado	Detalhes

1	Implementado (c/ limitações)	Impossibilidade de subscrever a mais que 4 *Characteristics* do ESP 32
2	Implementado	
3	Implementado	
4	Implementado	
5	Implementado	
6	Implementado	
7	Implementado	
8	Não Implementado	
9	Implementado	
10	Não Implementado	
11	Implementado	
12	Não Implementado	
13	Implementado (c/ limitações)	O mapa guardado não inclui as *layers* operacionais
14	Não Implementado	
15	Implementado	
16	Implementado	

Como se pode constatar na Tabela 8, a grande maioria dos requisitos propostos foram implementados havendo, no entanto algumas excepções.

O Requisito 1, ainda que implementado, apresenta algumas limitações. No teste da interface *BLE* do ESP32 verificou-se que haviam restrições no número de notificações de *Characteristics* a que se podia subscrever.

O teste foi feito tanto com a aplicação desenvolvida como com outra aplicação disponível na *Play Store* cujo propósito era apenas o de utilizar esta

interface. Em ambas as aplicações verificou-se que só se conseguia subscrever a um máximo de 4 notificações, número a partir do qual se constataram incongruências nos dados transmitidos fazendo com que certas *Characteristics* começassem a apresentar o valor de outras.

Dado que não foi descoberta a causa deste problema, optou-se por implementar uma estrutura de dados que permitisse transmitir todos os dados desejados utilizando no máximo 4 *Characteristics,* acabando-se por tomar decisões como a de juntar a latitude e longitude numa *Characteristic* e a temperatura e humidade noutra tendo-se depois feito o *parsing* na aplicação.

Para a implementação do Requisito 8, foram usadas as funções *area()* e *length()* da classe *GeometryEngine* que permitem calcular a área de um *Polygon* e o comprimento de uma *Polyline*, respetivamente.

Estas funções retornam um valor cujas unidades de medida não são especificadas, pelo que se recorreu ao *Google Maps* onde se criaram geometrias semelhantes para perceber quais eram as unidades de medida em que estavam essas geometrias.

Neste processo detetou-se que para uma geometria desenhada na aplicação desenvolvida em comparação com outra semelhante desenhada no *Google Maps,* a diferença dos valores das medidas obtidas eram muito significativas.

Como não se conseguiu descobrir a causa desta divergência entre valores considerou-se este requisito como "Não Implementado".

Para o Requisito 10 foram ponderadas duas estratégias de obtenção do tamanho da frente de fogo a partir de uma foto, ambas baseadas na equação **(1)**.

$$Dist.a\ objeto\ (mm) = \frac{dist_{focal}(mm) \times alt_{real}\ (mm) \times alt_{imagem}\ (px)}{alt_{objeto}\ (px) \times alt_{sensor}\ (mm)} \qquad \textbf{(1)}$$

Em que $dist_{focal}$ se refere à distância focal, alt_{real} à altura real de um determinado objeto, alt_{objeto} à altura do objeto em píxeis na imagem e alt_{sensor} à altura do sensor que capta a imagem.

As estratégias utilizadas são as seguintes:

- Cálculo com altura conhecida – esta estratégia implica saber atempadamente as dimensões reais de um determinado objeto presente na fotografia.
- Cálculo com tamanho estimado – esta estratégia implica fazer uma estimativa da altura de um determinado objeto na fotografia.

A primeira estratégia tem a desvantagem de se ter que saber o tamanho de um determinado objeto na fotografia. Este poderá ser um objeto introduzido cujas dimensões sejam conhecidas. Tal implica que, caso se opte por introduzir um objeto, se tenha que o fazer constantemente em todas as fotografias, o que pode claramente interferir com a operacionalidade do meio que tira a fotografia.

Por outro lado, a segunda estratégia implica que se faça uma estimativa empírica do tamanho de um objeto, por exemplo, do tamanho do fogo. Isto poderá introduzir erros significativos na determinação da distância a um ponto dado que a estimativa poderá divergir bastante da medida real.

Por fim, este requisito acabou por não ser implementado dado não ter uma importância elevada e, consequentemente, tenha havido um maior investimento de tempo na conclusão de outros requisitos.

No que toca ao Requisito 12, procurou-se dar-lhe resposta de forma a que a aplicação pudesse trabalhar 100% *offline* sem ter que recorrer a uma ligação à *Internet* para descarregar os mapas da *ESRI*.

Este não foi concluído dado que a criação de um *Tile Package* através do ArcGIS Pro (*software* que não está contemplado no *ArcGIS for Developers* mas que dispõe de uma licença *trial* para testar) limitava o nível de detalhe do mapa.

Assim sendo, optou-se por não concluir este requisito uma vez que os mapas-base obtidos poderiam limitar a operacionalidade dos meios no TO dado o seu nível de detalhe reduzido.

O Requisito 13 não foi desenvolvido por completo uma vez que no seu teste verificou-se que o mapa gravado na conta ArcGIS Online não contava com as *layers* operacionais.

Finalmente, o Requisito 14 foi testado com sucesso com um mapa criado no ArcGIS Online, descarregando a área de interesse para trabalhar *offline*. No entanto, dado que não se implementou o acesso ao *Portal ArcGIS* de onde se poderiam consultar e aceder aos mapas desenvolvidos na conta do *ArcGIS Online*, admitiu-se que este requisito não foi implementado.

4.3. Trabalho futuro

Com vista à melhoria do projeto desenvolvido nesta dissertação, são propostas algumas correções e desenvolvimentos a serem feitos.

No que diz respeito a correções, é apresentada de seguida uma lista com algumas sugeridas:

- Corrigir a limitação do ESP32 em apenas notificar até 4 *Characteristics*.
- Criar um maior desacoplamento entre componentes no código da aplicação *tablet*.
- Melhorar a obtenção da localização do *tablet*.
- Gerir a transição entre modos de exibição (*landscape* e *portrait*) guardando as variáveis de estado e recuperando o estado anterior.
- Corrigir o erro de não anexar as *layers* do mapa a ser guardado no ArcGIS Online.
- Corrigir o erro de não se poder eliminar múltiplos anexos de uma vez.

Por outro lado, os novos desenvolvimentos sugeridos são:

- Integrar mais sensores no *device* em particular sensores de direção e intensidade do vento.

- Disponibilizar novas funcionalidade de *SIG* como, por exemplo, pesquisar uma rota para um destino, obter a rota mais curta, pesquisar por conteúdo, entre outras.
- Desenvolver a obtenção da distância até à frente de fogo e sua consequente georreferenciação.
- Criação de *Tile Packages* que permitam à aplicação usar mapas-base armazenados localmente no *tablet* e assim não depender de conexão à Internet.

É também de se salientar que a colaboração de entidades que venham a beneficiar do sistema proposto é preponderante para o desenho de uma solução que responda às necessidades dos meios envolvidos no combate aos incêndios.

A integração de outras áreas científicas que estejam na esfera dos incêndios introduzirá conhecimento que contribuirá para o desenvolvimento de uma solução que permita um apoio à decisão mais informado e assim conduzir a uma resposta mais eficaz dos meios no TO.

Bibliografia

[1] Pordata.pt. 2019. PORDATA - Incêndios rurais e área ardida – Continente [online] Available at: Https://www.pordata.pt/Portugal/Inc%c3%aandios+rurais+e+%c3%a1rea+ardida+%e2%80%93+Continente-1192-9576 [Accessed 3 Mar. 2019]

[2] Martins, S. (2010). Incêndios Florestais: Comportamento, Segurança e Extinção. Mestrado Interdisciplinar em Dinâmicas Sociais, Riscos Naturais e Tecnológicos. Faculdade de Ciências e Tecnologia Departamento de Engenharia Mecânica.

[3] Beighley, M. and Hyde, A. (2018). Gestão dos Incêndios Florestais em Portugal numa Nova Era Avaliação dos Ris-cos de Incêndio, Recursos e Reformas. [online] p.26. Available at: https://www.isa.ulisboa.pt/files/cef/pub/articles/2018-04/2018_Portugal_Wildfire_Management_in_a_New_Era_Portuguese.pdf [Accessed 1 Mar. 2019].

[4] Autoridade Nacional de Proteção Civil (ANPC) (2019). Diretiva Operacional Nacional Nº2/2019 – Dispositivo Especial de Combate a Incêndios Rurais. Carnaxide: ANPC

[5] Autoridade Nacional de Proteção Civil (ANPC) (2019). Entrevista na ANPC sobre o atual dispositivo tecnológico e funcionamento do sistema de combate aos incêndios florestais

[6] Faden, R. (2017). Resposta à pergunta nº. 4560/XIII/2ª., de 23 de junho de 2017 - Incêndio florestal ocorrido em Pedrogão Grande, a 17 de junho. Lisboa: Gabinete do Primeiro-Ministro.

[7] Diário da República n.º 209/2017, Série I (2017-10-30)

[8] Particle, "Electron Datasheet v005", 2019

[9] Johnson, N. (2019). Opti stops floods before they start: with the power of Particle. [online] Available at: https://cdn2.hubspot.net/hubfs/2833873/Particle-CaseStudy-Opti%20(1).pdf [Accessed 6 Jan. 2019].

[10] E. Systems, “ESP32 Series Datasheet,” 2019.

[11] Esp32.net. (2019). The Internet of Things with ESP32. [online] Available at: http://esp32.net/ [Accessed 5 Mar. 2019].

[12] Pycom, “LoPy 4 Datasheet Version 1.0” 2019.

[13] Bipasha Biswas, S., & Tariq Iqbal, M. (2018). Solar Water Pumping System Control Using a Low Cost ESP32 Microcontroller. Em 2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE). IEEE.

[14] Raspberrypi.org. (2019). Raspberry Pi 3 Model B+ Product brief. [online] Available at: https://www.raspberrypi.org/documentation/faqs/ [Accessed 6 Jan. 2019].

[15] Raspberrypi.org. (2019). FAQs - Raspberry Pi Documentation. [online] Available at: https://www.raspberrypi.org/documentation/faqs/ [Accessed 6 Jan. 2019].

[16] Waheed, S. A., & Sheik Abdul Khader, P. (2017). A Novel Approach for Smart and Cost Effective IoT Based Elderly Fall Detection System Using Pi Camera. Em 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC). IEEE.

[17] Shah, V., Patel, R., & Nayak, R. (2017). Short Range Inter-satellite Link for Data Transfer and Ranging using IEEE802.11n. International Journal of Computer Applications, 164(1), 23–25.

[18] Shahzad, K., & Oelmann, B. (2014). A comparative study of in-sensor processing vs. raw data transmission using ZigBee, BLE and Wi-Fi for data intensive monitoring applications. Em 2014 11th International Symposium on Wireless Communications Systems (ISWCS).

[19] Pei, L., Liu, J., Guinness, R., Chen, Y., Kroger, T., Chen, R., & Chen, L. (2012). The evaluation of WiFi positioning in a Bluetooth and WiFi coexistence environment. Em 2012 Ubiquitous Positioning, Indoor Navigation, and Location Based Service (UPINLBS). IEEE.

[20] Mikhaylov, K. (2014). Simulation of network-level performance for Bluetooth Low Energy. Em 2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC). IEEE.

[21] Gomez, C., Oller, J., & Paradells, J. (2012). Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. Sensors, 12(9), 11734–11753.

[22] Android Developers. (2019). Bluetooth low energy overview | Android Developers. [online] Available at: https://developer.android.com/guide/topics/connectivity/bluetooth-le [Accessed 7 Jan. 2019].

[23] Bluetooth.com. (2019). GATT Overview | Bluetooth Technology Website. [online] Available at: https://www.bluetooth.com/specifications/gatt/generic-attributes-overview [Accessed 7 Jan. 2019].

[24] Sirur, S., Juturu, P., Gupta, H. P., Serikar, P. R., Reddy, Y. K., Barak, S., & Bonggon Kim. (2015). A mesh network for mobile devices using Bluetooth low energy. Em 2015 IEEE SENSORS. IEEE.

[25] Noreen, U., Bounceur, A., & Clavier, L. (2017). A study of LoRa low power and wide area network technology. Em 2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP). IEEE.

[26] Zhang, X., Zhang, M., Meng, F., Qiao, Y., Xu, S., & Hour, S. H. (2018). A Low-Power Wide-Area Network Information Monitoring System by Combining NB-IoT and LoRa. IEEE Internet of Things Journal, 1–1.

[27] Sherazi, H. H. R., Imran, M. A., Boggia, G., & Grieco, L. A. (2018). Energy Harvesting in LoRaWAN: A Cost Analysis for the Industry 4.0. IEEE Communications Letters, 22(11), 2358–2361.

[28] Cdn.rohde-schwarz.com. (2019). Narrowband Internet of Things Whitepaper.

[online] Available at: https://cdn.rohde-schwarz.com/pws/dl_downloads/dl_application/application_notes/1ma266/1MA266_0e_NB_IoT.pdf [Accessed 8 Jan. 2019].

[29] Vejlgaard, B., Lauridsen, M., Nguyen, H., Kovacs, I. Z., Mogensen, P., & Sorensen, M. (2017). Coverage and Capacity Analysis of Sigfox, LoRa, GPRS, and NB-IoT. Em 2017 IEEE 85th Vehicular Technology Conference (VTC Spring). IEEE.

[30] Oh, J., & Song, H. (2018). Study on the Effect of LTE on the Coexistence of NB-IoT. Em 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN). IEEE

[31] S. Sharma, D. Kumar and K. Kishore, "Wireless Sensor Networks-A Review on Topologies and Node Architecture", International Journal of Computer Sciences and Engineering, Vol.1, Issue. 2, 2013

[32] Ghimire L. (2015) Joint analysis of packet size and forward error correction in IEEE 802.15.4 type networks. University of Oulu, Department of Communications Engineering, Master's Degree Programme in Wireless Communications Engineering, 88 p.

[33] Abdullah, S., Bertalan, S., Masar, S., Coskun, A., & Kale, I. (2017). A wireless sensor network for early forest fire detection and monitoring as a decision factor in the context of a complex integrated emergency response system. Em 2017 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS). IEEE.

[34] Zhu, Y., Xie, L., & Yuan, T. (2012). Monitoring system for forest fire based on wireless sensor network. Em Proceedings of the 10th World Congress on Intelligent Control and Automation. IEEE.

[35] Devadevan, V., & Suresh, S. (2016). Energy Efficient Routing Protocol in Forest Fire Detection System. Em 2016 IEEE 6th International Conference on Advanced Computing (IACC). IEEE.

[36] Earthobservatory.nasa.gov. (n.d.). A Clearer View of Fire. [online] Available at: https://earthobservatory.nasa.gov/images/87111/a-clearer-view-of-fire [Accessed 11 Jan. 2019].

[37] Maden, U. (2018). Satellite data aids forest fire detection and monitoring in Nepal. [online] Phys.org. Available at: https://phys.org/news/2018-06-satellite-aids-forest-nepal.html#jCp [Accessed 12 Jan. 2018].

[38] Kiadtikornthaweeyot, W., Sukawattanavijit, C., & Rungsipanich, A. (2018). Automatic detection of forest fire burnt scar from Landsat-8 image of northern part of Thailand. Em 2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON). IEEE.

[39] European Space Agency. (n.d.). Overview. [online] Available at: https://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Overview3 [Accessed 27 Jun. 2019].

[40] Copernicus.eu. (n.d.). About Copernicus. [online] Available at:

https://www.copernicus.eu/en/about-copernicus [Accessed 27 Jun. 2019].

[41] Dgterritorio.pt. (2017). DGTerritório - Copernicus. [online] Available at: http://www.dgterritorio.pt/copernicus/ [Accessed 27 Jun. 2019].

[42] Incêndios Florestais Modelação Aplicada. (2017). PROCIV, [online] (88), p.5. Available at: http://www.prociv.pt/bk/newsletter/ProcivN%C2%BA88.pdf [Accessed 27 Jun. 2019].

[43] Barreirinha, P. (2019). Entrevista ao Comandante dos Bombeiros Voluntários de Ílhavo, Pedro Barreirinha sobre o funcionamento do combate aos incêndios florestais

[44] Duan, Y. X., Cao, J. Z., & Luo, Z. L. (2011). Intelligent GIS system of forest fire alarm and it's controlling strategy design. Em 2011 International Conference on Machine Learning and Cybernetics. IEEE.

[45] GIS Geography. (2019). What is Geographic Information Systems (GIS)? - GIS Geography. [online] Available at: https://gisgeography.com/what-gis-geographic-information-systems [Accessed 9 Jan. 2019].

[46] Fonseca, M. (2018). Mação | António Louro, o senhor MacFire. Mediotejo.net. [online] Available at: http://www.mediotejo.net/macao-antonio-louro-o-senhor-macfire/ [Accessed 20 Feb. 2019].

[47] Carvalho, F. (2015). Sistema de apoio no combate aos fogos florestais para o Concelho de Águeda. Mestrado em Geoinformática. Universidade de Aveiro.

[48] Viegas, D. (2017). O complexo de incêndios de Pedrógão Grande e concelhos limítrofes, iniciado a 17 de junho de 2017. [online] Coimbra: Centro de Estudos sobre Incêndios Florestais, p.205. Available at: https://cdn.cmjornal.pt/files/2017-12/2017-12-07_15_54.38_Relat_rio_fogos_Xavier_Viegas_aaa.pdf [Accessed 20 Feb. 2019].

[49] Expresso (2018). Combate aos fogos com nova arma no terreno. [online] Available at: https://www.fct.unl.pt/sites/default/files/documentos/noticias/2018/expresso_18_8.pdf [Accessed 20 Feb. 2019].

[50] Esri_Portugal (2015). Plataforma Integrada de Gestão de Riscos (PGIR). Available at: http://www.esriportugal.pt/Inovacao-PGIR [Accessed 27 Jun. 2019].

[51] Projecto GeoMAI. [video] Available at: https://www.youtube.com/watch?v=Nb4hj3HET9c [Accessed 27 Jun. 2019].

[52] Secretaria Geral do MAI. (2014). Sistemas de Informação. [online] Available at: https://www.sg.mai.gov.pt/Tecnologias/SistemasInformacao/Paginas/default.aspx [Accessed 27 Jun. 2019].

[53] Diário da República n.º 143/2017, Série I (2017-07-26)

[54] Copernicus.eu. (n.d.). Emergency | Copernicus. [online] Available at: https://www.copernicus.eu/en/services/emergency [Accessed 27 Jun. 2019].

[55] Emergency.copernicus.eu. (n.d.). Copernicus Emergency Management Service. [online] Available at: https://emergency.copernicus.eu/ [Accessed 27 Jun. 2019].

[56] Effis.jrc.ec.europa.eu. (n.d.). EFFIS - Welcome to EFFIS. [online] Available at: http://effis.jrc.ec.europa.eu/ [Accessed 27 Jun. 2019].

[57] Copernicus.eu. (n.d.). European Forest Fire Information System | Copernicus. [online] Available at: https://www.copernicus.eu/en/european-forest-fire-information-system [Accessed 27 Jun. 2019].

[58] Copernicus EU (2017). Copernicus Emergency Management Service: Forecasting forest fires with EFFIS. [video] Available at: https://www.youtube.com/watch?v=s_rymArjxCc [Accessed 27 Jun. 2019].

[59] Ticnologia.pt. (2010). Indra fecha contrato de 450 mil Euros com Autoridade Nacional de Protecção Civil. [online] Available at: http://ticnologia.pt/noticias/negocios/indra-fecha-contrato-de-450-mil-euros-com-autoridade-nacional-de-proteccao-civil.html?eprivacy=1 [Accessed 27 Jun. 2019].

[60] Segurança Online. (2011). SADO: a nova plataforma para troca de informação entre os agentes da ANPC. [online] Available at: http://www.segurancaonline.com/noticias/detalhes.php?id=170 [Accessed 27 Jun. 2019].